(2)

FJSRL-TR-91-0002

FRANK J. SEILER RESEARCH LABORATORY

A 3-DEGREE-OF-FREEDOM FLIGHT SIMULATOR EVALUATION OF UNSTEADY AERODYNAMICS EFFECTS

LT COL C. BRUCE HARMON
MAJOR WILLIAM DIETERICH

AUGUST 1991

AIR FORCE SYSTEMS COMMAND

UNITED STATES AIR FORCE

FJSRL–TR–91–0002

This document was prepared by the Aerospace Sciences Division, Frank J. Seiler Research Laboratory, United States Air Force Academy, CO. The research was conducted under Project Work Unit Number 2307–F1–38, Unsteady Airfoil Energized Flow. Lt Colonel Chester B. Harmon was the Project Scientist in charge of the work.

Inquiries concerning the technical content of this document should be addressed to the Frank J. Seiler Research Laboratory (AFSC), FJSRL/NA, USAF Academy, CO 80840–6528. Phone (719) 472–2812.

This technical report has been reviewed and is approved for publication.

[illegible]
CHESTER B. HARMON,
Lt Col, USAFR
Research Associate

[illegible]
RICHARD W. NEWSOME, Jr.,
Lt Col, USAF
Chief, Aerospace Sciences Division

[illegible]
BARRY G. MORGAN, Lt Col, USAF
Commander

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE August 1991	3. REPORT TYPE AND DATES COVERED Technical Report

4. TITLE AND SUBTITLE

A 3–Degree–of–Freedom Flight Simulator Evaluation of Unsteady Aerodynamics Effects

5. FUNDING NUMBERS

2307–F1–38

6. AUTHOR(S)

C. Bruce Harmon
William Dieterich

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Frank J. Seiler Research Laboratory
USAF Academy CO 80840–6528

8. PERFORMING ORGANIZATION REPORT NUMBER

FJSRL–TR–91–0002

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report documents the delivery of a 3–degree–of–freedom flight simulator for the use of researchers in the field of unsteady aerodynamics.

14. SUBJECT TERMS

Unsteady aerodynamics;
Flight simulation

15. NUMBER OF PAGES

88

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	NONE

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page Instructions for filing in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*

Block 2. Report Date. Full publication date including day, month, and year, if available (e g 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88)

Block 4. Title and Subtitle A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s) Use the following labels:

C -	Contract	PR -	Project
G -	Grant	TA -	Task
PE -	Program Element	WU -	Work Unit Accession No.

Block 6. Author(s) Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s)

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number Enter the unique alphanumeric report number(s) assigned by the organization performing the report

Block 9 Sponsoring/Monitoring Agency Name(s) and Address(es) Self explanatory

Block 10 Sponsoring/Monitoring Agency Report Number *(If known)*

Block 11 Supplementary Notes Enter information not included elsewhere such as. Prepared in cooperation with , Trans of , To be published in When a report is revised, include a statement whether the new report supersedes or supplements the older report

Block 12a. Distribution/Availability Statement Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE -	See authorities.
NASA -	See Handbook NHB 2200.2
NTIS -	Leave blank

Block 12b. Distribution Code

DOD -	Leave blank.
DOE -	Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA -	Leave blank
NTIS -	Leave blank

Block 13. Abstract Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code Enter appropriate price code *(NTIS only)*

Blocks 17. - 19. Security Classifications Self-explanatory. Enter U S Security Classification in accordance with U S Security Regulations (i e , UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page

Block 20. Limitation of Abstract This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited If blank, the abstract is assumed to be unlimited

A 3-Degree-of-Freedom Flight Simulator for Evaluation of Unsteady Aerodynamics Effects

FINAL REPORT

Lt Col C Bruce Harmon
Maj William Dieterich

Frank J. Seiler Research Lab
US Air Force Academy, Colorado

30 August 1991

ACKNOWLEDGEMENT

The authors would like to thank Capt Scott Schreck for his advice and encouragement throughout this endeavor.

INTRODUCTION

This report documents the delivery of a 3-degree-of-freedom flight simulator for the use of researchers in the field of unsteady aerodynamics. It was accomplished by Lt Col Harmon and Major Dieterich for AFOSR at the FJ Seiler Research Lab during tours of duty in 1990 and 1991. Both are assigned AF Reservists.

The software is distributed to qualified users via 3-1/2 inch high density diskette. A qualified user is a US Government agency or a contractor under US Government contract. Contact Capt Scott Schreck at the Frank J. Seiler Research Lab at 719-472-2812 if you think you qualify. There will be a nominal charge for the media and handling.

This report, as well as the supporting documentation, is also distributed on that diskette as a set of ASCII text files so that it may be kept on line in the same directory with the simulation and its files. This file is named report.doc. The appendices are stored as appX.doc, where X is the designation of that appendix. Directory of appendices follows:

A	appa.doc	documents differences between MSDOS and VMS
B	appb.doc	data dictionary
C	appc.doc	example input and output files
D	appd.doc	flight sim source files
E	appe.doc	documentation and sources for 3-d graphics back-end
F	appf.doc	example input and output for 3-d graphics back-end
G	appg.doc	documentation and sources for 2-d graphics back-end
H	apph.doc	example input and output for 2-d graphics back-end
I	appi.doc	sources for Ada flight sim
J	appj.doc	test cases documentation

Note that the electronic copy of appendices F and H does not include the output since these programs write to the screen, a laser printer, or a plotter. The hardcopy final report does contain output.

The flight simulation was designed to be portable across programming languages and computer operating systems. It was written in the C programming language under MSDOS and subsequently ported to VAX/VMS, first in C then in Ada. The input files are identical under both operating systems and both languages. The output files are identical across these environments except for minor differences under Ada. The C source files are different only in their syntax for the use of include files. The build (make) files are different (see file appa.doc) as are the binary object and executables. The documentation files are identical.

The MSDOS version was compiled using Turbo C++ 1.0 under MSDOS 4.01 on the Unisys Desktop III computer, a 386 machine. However, the source remains a proper C subset of C++. The VMS version was built using the standard DEC (Digital Equipment Corportation) C compiler under VMS 5.41. To ensure portability, the new features of ANSI C (such as function prototypes) were not used. They are not permitted under DEC's VAX/VMS C. To permit ease of porting to other languages, the data types and program structures were constrained to the basic types. An Ada version of the simulation is provided as an appendix. A port to Fortran or Pascal was conducted but not included in the file set. A port to Unix using the Hewlett Packard HP9000 Series 300 was also conducted but is not provided with the file set.

The build scripts for MSDOS and VMS are provided on line as an appendix under file appa.doc.

Under MSDOS the simulation uses a standard-in, standard-out i/o approach. That is, one file is read, and one is written. The same approach applies to VAX/VMS although the terminology of standard-in, standard-out does not apply. The approach would be the same for Unix systems. Here is the command line syntax for MSDOS (fs <fs1.in >fs1.out) and VMS (run /input=fs1.in /output= fs1.out fs), where fs1.in is the input and fs1.out is the output.

In addition, the authors have built a set of graphics "back ends" for MSDOS and VMS applications. These take the raw output after appropriate filtering and draw graphical output on the computer screen. Under MSDOS, the Borland graphics of Turbo C++ is used to paint two-dimensional results on EGA and VGA screens. A dump function is provided as well. This dump function allows the screen dump from an EGA or VGA to an HP Laserjet or Deskjet via PCL commands. Under VAX/VMS these back end programs are based on Fortran and Disspla software and paint Tektronix graphics terminals with dump capability to HP 7550 plotters. These back ends are provided as appendices to this report.

Before these graphics back ends can be used, the flight simulator output file must be put in the proper form. The graphics input files must have a header record followed by records of two or three floating point numbers. A description of these transformations is given in the appendix associated with the back ends.

TOP LEVEL VIEW

The user would do well to print out the text files in this directory so that cross referencing is facilitated. Naturally, a Windows-based machine helps a great deal.

The directory contains several files whose extension names provide clues:

.c	C source files fs.c and mv.c fs contains the main program, mv math functions
.obj	corresponding object files
.exe	the executable image file fs.exe
.mak	makefile; used to "build" the program using utility "make"
.in	input file; ASCII text format
.out	output file; ASCII text format
.doc	documentation

A user does not have to bother with most of these files, but they help the person who intends to modify the simulation. For example, to modify the simulation, use any text editor to change the sources and then type "make -ffs.mak" at the prompt. All users should read every file with the .doc extension before beginning to use the simulation.

The simulation is built on a standard-in standard-out i/o model. That means that only one input file is read and only one output file is written. Both are ASCII text files. This approach enhances portability across platforms, languages, and operating systems. Further, standard Unix filters for text processing can be used. Under MSDOS to execute the fs program using fs1.in as input and fs1.out as output, just type "fs <fs1.in >fs1.out" at the prompt. The < sign "redirects" the standard input to come from file fs1.in, while the > sign redirects the standard output to go to fs1.out. If fs1.out does not exist, it is created. If it does exist, it is destroyed and rewritten with new contents.

Filters and editors can be applied to the resulting output file to put it into a form for display on the screen or for sending to a plotter or laser printer. But the raw output is a simple ASCII text file that echos the inputs then writes line-oriented records for each time step the simulation takes. This file can be easily viewed at the terminal by using the "type <filename>" command.

DATA DICTIONARY

All variables used in the simulation are documented in a data dictionary, contained on line as an appendix in file appb.doc.

THE SIMULATION, A TECHNICAL VIEW

The simulation responds to three independent controls. The "pilot" sets a thrust value that is valid for the duration. The bank angle is constrained to its initial value for the duration. The pitch is controlled through a CL vs time profile "commanded" by the pilot in accordance with the input profile. As an alternative, the simulation can respond to a constant airspeed profile. In that case, the thrust will vary as required to maintain that airspeed. Constant bank angle and CL vs time are still applied as before.

The simulation begins by reading the appropriate simulation control parameters, the aerodynamic parameters, and then the initial conditions. These numbers are used to calculate other initial conditions and the results are echoed to the output file. The simulation then enters the main loop at the initial time. The forces acting on the aircraft are calculated. Lift is determined from the input file coefficient of lift commanded by the pilot. Drag is a consequence of that lift and the airspeed. Thrust is commanded by the pilot. These three forces are expressed in the body stability axis system and resolved with the weight vector, which is expressed in the inertial earth system. To accomplish this, the initial Euler angles are used for the first calculation. Subsequently, the Euler angles are calculated with each time step and used to resolve the forces with each time step as well.

The Euler angles are used to determine the transformation matrices to resolve the vectors that must be added to each other to form resultant forces. These forces produce the corresponding accelerations via Newton's F=ma. They are then transformed to the inertial earth axis system and integrated for velocity and position updates. The inertial velocity vector is then used to determine the new Euler angles. This allows the rate of change of the Euler angles to be determined via (new - old)/dt. Then these rates are transformed to produce the stability axis rotations p,q,r. Then the time is stepped by a discrete dt and control goes to the top of the main loop.

The derivations of the EOMs are found in Roskam(1) and Greenwood(2), primarily. The methodology of the simulation is patterned after the TAC Zinger series, developed at USAF Studies and Analysis circa 1975. Numerical integration is via the rectangle method. This method is the simplest to implement and is more stable than trapezoid, Runge Kutta, etc. We chose this method, planning to go to other, more sophisticated methods if the simulations took too long. This simulation can be run for 100.0 seconds of simulation time using dt=0.01 sec in less than 25 real seconds under MSDOS on the Unisys Desktop III, a 386 machine. Most of the unsteady phenomena are captured in less than 5 seconds simulation time.

The input file contains the provision to load in coefficient of lift versus time. This is the unsteady aerodynamics "hook". Since much of the research in this field is oriented toward enhanced lift for rapid turning and pointing applications, the researcher is encouraged to develop a CL versus time profile and use it for input to the model. The research suggests that these effects are very short in duration but produce CL well above the steady state values. Other users can just provide a few points of constant CL versus time.

SIMPLIFICATIONS

The EOM for the general case of six-degree-of-freedom rotating and translating rigid bodies are non-linear and can be simplified to facilitate simulation. Several simplifications were made to these EOM that are documented here.

The model constrains phidot to zero. That is, the initial angle of bank is fixed for the duration. This constraint was imposed since the pitch axis is the important one for unsteady effects.

Further, there are no moment of interia effects. That is, the Qdot is not contrained by Iyy. The aircraft is permitted to align the X axis with the flight path exactly. The small effects of alpha (thrust not exactly on centerline with relative wind) are ignored. The body axis chosen is the stability axis, where the X axis is exactly tangent to the flight path. Thus U is exactly airspeed; V = W = 0.

The EOM used then are from Roskam eq 2-19, simplified. The following code fragment illustrates.

```
/* using eq 2-19, compute linear accel (f=ma), stab axes */
```

```
        udots[0] = (wts[0] - drag + thrust) / mass;
        udots[1] =  wts[1] / mass;
        udots[2] = (wts[2] - lift) / mass;
/* if constant airspeed profile, then set tangential accel to zero */
        if (iconas) udots[0] = 0.0;
/* airspeed change is entirely due to udots[0]          */
        aspd = aspd + dt * udots[0];
/* convert stability udots vector to inertial udot1 */
        mvmult(aphi, udots,udot3);
        mvmult(athet,udot3,udot2);
        mvmult(apsi, udot2,udot1);
/* advance xdot1 then x1 */
        for (i=0; i<3; i++) {
            xdot1[i] = xdot1[i] + dt * udot1[i];
            x1[i]    = x1[i]    + dt * xdot1[i];
        }
/* geometry to get psi, thet (phi constrained to its initial value) */
        psi = atan( xdot1[1] / xdot1[0] );
        if ( xdot1[0] < 0.0 )  psi = psi + PI;
          else if ( xdot1[1] < 0.0 ) psi = psi + TWOPI;
        psidot  = (psi  - psib)  / dt;
        if ( (psi-psib) > PI ) psidot = 0.0;
        if ( (psib-psi) > PI ) psidot = 0.0;

        temp = sqrt(xdot1[0]*xdot1[0] + xdot1[1]*xdot1[1]);
        thet = atan( -xdot1[2] / temp );
        thetdot = (thet - thetb) / dt;
        phidot  = 0.0;    /* pilot constraint */
/* use eq 2-46 to get p,q,r from psidot, thetdot, phidot    */
        p =  phidot - psidot * sin (thet);
        q =  thetdot * cos(phi) + psidot * cos(thet) * sin(phi);
        r = -thetdot * sin(phi) + psidot * cos(thet) * cos(phi);
        omegas = sqrt(p*p + q*q + r*r);  /* magnitude */
        omegae = sqrt(psidot*psidot + thetdot*thetdot + phidot*phidot);
```

The first block of code computes the linear acceleration in the stability (body) axes from the forces acting on the airplane. Subscripts 0,1,2 represent x,y,z respectively. These new linear accelerations are transformed to inertial axes via the mvmult function calls. Inertial velocity and position are updated via numerical integration (rectangle rule). Then geometry is used to derive the new Euler angles from the inertial velocity vector. Then (new - old)/dt is used to determine the rate of change of each of the Euler angles. Then Roskam equation 2-46 is used to compute the p,q,r rotation rates (stability axis). Omegas and omegae should be equal since they are both magnitudes of the same vector expressed in two different coordinate systems.

REFERENCES

1. Roskam, Jan, Airplane Flight Dynamics and Automatic Flight Controls, Roskam Aviation and Engineering Corporation, 1979, pp 9-51.

2. Greenwood, Donald T., Principles of Dynamics, Prentice Hall, 1965, pp 38-40 and pp 281-336.

APPENDIX A

APPENDIX B

APPENDIX C

```
This file, appa.doc, documents the differences between
the MSDOS and VMS flight simulation sources and their
build files.
-----------------------------------------------------------
Re C source files, on the VAX:
#include stdio
#include math

On MSDOS under Turbo C++:
#include <stdio.h>
#include <math.h>
-----------------------------------------------------------
Re makefiles, on the VAX:
cc fs
link fs,mv
run /input=fs.in /output=fs.out fs

On MSDOS under Turbo C++: make -ffs.mak, where fs.mak is:
# file=fs.mak              make -ffs.mak
# Note, C++ may give warnings since ANSI conventions are not followed.

MODEL  = m
OBJS   = fs.obj mv.obj
LIBS   = \tc\lib\emu \tc\lib\math$(MODEL) \tc\lib\c$(MODEL)
CFLAGS = -c -m$(MODEL) -I\tc\include

fs.exe: $(OBJS)
  tlink \tc\lib\c0$(MODEL) fs mv, $*, ,$(LIBS)

.c.obj:
  tcc $(CFLAGS) $<
```

```
DATA DICTIONARY, file=appb.doc
This file contains the definitions of variables used in the fs
flight simulator.

GLOBALS

p,q,r = roll, pitch, yaw rates (radians/sec)
psi, thet, phi = Euler angles  (radians)
psidot, thetdot, phidot = time derivatives of Euler angles
omegae = magnitude of psidot,thetdot,phidot vector
omegas = magnitude of p,q,r vector; omegas should = omegae
t = simulation time (seconds)
tmin = minimum value of t
tmax = maximum value of t
dt   = simulation increment in t
apsi, athet, aphi = rotation matrices for converting a vector from
      inertial to body axes
aipsi, aithet, aiphi = matrix inversions of apsi ...
cd0 = coefficient of drag at lift = 0
aspect = aspect ratio, span/chord
eff    = Oswald's efficiency factor, used in cd calculation
aspd  = airspeed (ft/s)
alt   = altitude (ft)
thrust = engine thrust (pounds)
drag   = aircraft drag (pounds)
lift   = aircraft lift (pounds)
mass   = aircraft mass (slugs)
rho    = air density (slugs/ft3)
qbar   = dynamic pressure (pounds/ft2)
swing  = wing planform area (ft2)
tclmin = minimum time in cl vs time array, tclary[0]
tclmax = maximum time in cl vs time array, tclary[nptscl-1]
nptscl = number of points in the cl vs time array
tclary = an array of times indexed 0 .. nptscl-1
clary  = an array of cl    indexed 0 .. nptscl-1
dtcl   = time increment between each tclary[i]

wt, x, xdot, udot = vectors that express weight, position,
    velocity, acceleration.
    (Roskam notation convention)
    1 = inertial
    2 = rotated thru psi
    3 = rotated thru thet
    s = rotated thru phi (body or stability axis now)
    Thus wt1 is weight vector expressed in inertial
         wts is weight vector expressed in stability (body)
    x1 is position of aircraft (inertial)
    xdot1 is velocity of aircraft (inertial)
    udots is acceleration of aircraft (stability)
    udot1 is acceleration of aircraft (inertial)
Vectors and Arrays are indexed 0..2 in C; 1..3 in Ada, Fortran, Pascal.

idebug  = 0 normally, 1 if debug diagnostics should be printed
iconas  = 0 normally, 1 if const aspd profile is to be flown (that
          is, that the pilot will command whatever thrust is needed)

DTR    = conversion from degrees to radians 57.3
EE     = Euler's number
PI     = ratio of a circle's circumference to its diameter
TWOPI  = twice PI
```

An example input file is shown below.

```
0.0      60.0001  0.01   100   0   0
0.018    6.0   0.85
500.0    36000.0   4949.5647
10000.0   500.0    0.0   0.0    1.047198
0.0 500.0 2
0.672718
0.672718
```

Line 1 is expected to be composed of three floating point numbers followed by three integers. The first number is the time (sec) that the simulation begins, usually zero. The second number is the time that it ends. The third is the time step. The time step should be sufficiently small that changes in time step converge to the same results. The first integer is the print interval. That is, in this example, an output record is written every 100 time steps. The second integer is the debug flag. Set it to zero for normal behavior. The authors have placed diagnostic print statements in the code for other values of the debug flag. The third integer is iconas, another flag. Normally, it is set to zero. However, if it is set to one, the simulation will cause a constant airspeed manuever. Effectively, this means that the pilot is adjusting the thrust to maintain airspeed.

Line 2 is expected to contain three floating point numbers. The first one is the coefficient of drag at lift=0. The second is the aspect ratio, the ratio of the wing's span to chord. The third is the Oswald efficiency factor, used in the equation for the drag due to lift. It ranges from zero to one but is typically near 0.8.

Line 3 is expected to contain three floating point numbers. The first is the area of the wing in square feet. The second is the weight of the airplane in pounds. The third is the airplane thrust in pounds. These parameters are treated as constant by the simulation. The only exception to that is when the flag iconas is set to one, in which case the thrust is allowed to vary to permit constant airspeed flight.

Line 4 is expected to contain five floating point numbers. The first is the initial airplane altitude in feet. The second is the initial airspeed in feet per second. The last three are the initial Euler angles psi, thet, and phi (heading, pitch, and roll angles respectively) in radians.

Line 5 contains the parameters that are used to interpret the remaining lines. It contains two floating point numbers and an integer. The first number is the initial time for the array of coefficients of lift. The second is the time interval between each element of that array. The third number is an integer and specifies the number of array elements. Thus, in the example here, there will follow two elements of an array of coefficients of lift. The first element will correspond to time=0, while the second will correspond to time=500 sec. While simulation performs linear interpolation to ascertain the coefficient of lift during execution.

Thus, the remaining lines are two in number and specify the array of coefficients of lift described above.

Below is the output file that results from our example.

```
tmin,tmax,dt,iprint,idebug,iconas=    0.0000    60.0001    0.0100 100    0    0
```

```
cd0,aspect,eff=  0.0180   6.0000   0.8500
swing,weight,thrust=         500        36000           4949.5647
alt,aspd,psi,thet,phi=       10000      500   0.0000   0.0000   1.0472
mass,rho,qbar=   1118.92     0.00171246     214.0569
    0.00   500.00      2
    0.00   0.672718
  500.00   0.672718
apsi,athet,aphi
      1.00     -0.00      0.00
      0.00      1.00      0.00
      0.00      0.00      1.00

      1.00      0.00      0.00
      0.00      1.00      0.00
     -0.00      0.00      1.00

      1.00      0.00      0.00
      0.00      0.50     -0.87
      0.00      0.87      0.50

aipsi,aithet,aiphi
      1.00      0.00     -0.00
     -0.00      1.00     -0.00
      0.00     -0.00      1.00

      1.00     -0.00      0.00
     -0.00      1.00     -0.00
      0.00     -0.00      1.00

      1.00     -0.00     -0.00
     -0.00      0.50      0.87
      0.00     -0.87      0.50

wt1,wts,xdots,xdot1
      0.00
      0.00
  36000.00

      0.00
  31176.92
  17999.99

    500.00
      0.00
      0.00

    500.00
      0.00
      0.00

Output from program fs, flight sim for unsteady

   time       x      y      z aspd xdot ydot zdot  psi thet  phi   lift  drag

   0.00       0      0 -10000  500  500    0    0    0    0   60  72000  4950
   1.00     499     28 -10000  500  497   56    0    6   -0   60  72000  4950
   2.00     992    112 -10000  500  488  111    0   13   -0   60  72000  4950
   3.00    1472    249 -10000  500  472  164    0   19   -0   60  72000  4950
   4.00    1934    440 -10000  500  451  216    0   26   -0   60  72000  4950
   5.00    2373    680 -10000  500  425  264    0   32   -0   60  72000  4950
```

```
    6.00    2781     968  -10000   500   392   310    0    38   -0    60  72000   4950
    7.00    3156    1299  -10000   500   356   352    0    45   -0    60  72000   4950
    8.00    3491    1670  -10000   500   314   389    0    51   -0    60  72000   4950
    9.00    3783    2077  -10000   500   269   422    0    57   -0    60  72000   4950
   10.00    4027    2512  -10000   500   221   449    0    64   -0    60  72000   4950
   11.00    4222    2973  -10000   500   169   471    0    70   -0    60  72000   4950
   12.00    4365    3452  -10000   500   116   487    0    77   -0    60  72000   4950
   13.00    4453    3945  -10000   500    61   497    0    83   -0    60  72000   4950
   14.00    4486    4444  -10000   500     6   500    0    89   -0    60  72000   4950
   15.00    4464    4943  -10000   500   -50   498    0    96   -0    60  72000   4950
   16.00    4386    5437  -10000   500  -105   489    0   102   -0    60  72000   4950
   17.00    4253    5920  -10000   500  -159   475    0   109   -0    60  72000   4950
   18.00    4068    6385  -10000   500  -211   454    0   115   -0    60  72000   4950
   19.00    3833    6826  -10000   500  -260   428    0   121   -0    60  72000   4950
   20.00    3549    7238  -10000   500  -306   396    0   128   -0    60  72000   4950
   21.00    3222    7617  -10000   500  -348   360    0   134   -0    60  72000   4950
   22.00    2855    7957  -10000   500  -386   319    0   140   -0    60  72000   4950
   23.00    2452    8254  -10000   500  -419   274    0   147   -0    60  72000   4950
   24.00    2018    8504  -10000   500  -447   226    0   153   -0    60  72000   4950
   25.00    1560    8705  -10000   500  -469   175    0   160   -0    60  72000   4950
   26.00    1082    8853  -10000   500  -486   122    0   166   -0    60  72000   4950
   27.00     590    8947  -10000   500  -496    67    0   172   -0    60  72000   4950
   28.00      91    8987  -10000   500  -501    12    0   179   -0    60  72000   4950
   29.00    -409    8971  -10000   500  -499   -44    0   185   -0    60  72000   4950
   30.00    -905    8899  -10000   500  -491   -99    0   191   -0    60  72000   4950
   31.00   -1389    8772  -10000   500  -477  -153    0   198   -0    60  72000   4950
   32.00   -1857    8593  -10000   500  -457  -205    0   204   -0    60  72000   4950
   33.00   -2301    8363  -10000   500  -432  -254    0   211   -0    60  72000   4950
   34.00   -2718    8085  -10000   500  -401  -301    0   217   -0    60  72000   4950
   35.00   -3101    7762  -10000   500  -365  -343    0   223   -0    60  72000   4950
   36.00   -3446    7399  -10000   500  -325  -382   -0   230    0    60  72000   4950
   37.00   -3748    7000  -10000   500  -280  -415   -0   236    0    60  72000   4950
   38.00   -4005    6569  -10000   500  -232  -444   -0   242    0    60  72000   4950
   39.00   -4212    6113  -10000   500  -182  -467   -0   249    0    60  72000   4950
   40.00   -4367    5637  -10000   500  -129  -484   -0   255    0    60  72000   4950
   41.00   -4468    5146  -10000   500   -74  -496   -0   261    0    60  72000   4950
   42.00   -4514    4647  -10000   500   -19  -501   -0   268    0    60  72000   4950
   43.00   -4505    4146  -10000   500    37  -500   -0   274    0    60  72000   4950
   44.00   -4440    3649  -10000   500    92  -493   -0   281    0    60  72000   4950
   45.00   -4320    3163  -10000   500   146  -480   -0   287    0    60  72000   4950
   46.00   -4147    2692  -10000   500   199  -460   -0   293    0    60  72000   4950
   47.00   -3923    2244  -10000   500   248  -436   -0   300    0    60  72000   4950
   48.00   -3651    1823  -10000   500   295  -405   -0   306    0    60  72000   4950
   49.00   -3334    1435  -10000   500   338  -370   -0   312    0    60  72000   4950
   50.00   -2975    1085  -10000   500   377  -330   -0   319    0    60  72000   4950
   51.00   -2580     776  -10000   500   412  -287   -0   325    0    60  72000   4950
   52.00   -2153     513  -10000   500   441  -239   -0   332    0    60  72000   4950
   53.00   -1700     300  -10000   500   465  -189   -0   338    0    60  72000   4950
   54.00   -1225     137  -10000   500   483  -136   -0   344    0    60  72000   4950
   55.00    -736      28  -10000   500   495   -82   -0   351    0    60  72000   4950
   56.00    -237     -25  -10000   500   501   -26   -0   357    0    60  72000   4950
   57.00     264     -24  -10000   500   501    29   -0     3    0    60  72000   4950
   58.00     762      34  -10000   500   495    85   -0    10    0    60  72000   4950
   59.00    1251     146  -10000   500   482   139   -0    16    0    60  72000   4950
   60.00    1725     312  -10000   500   464   192   -0    22    0    60  72000   4950
```

That was the end of the output file. As you can see, the first output lines just echo the input. Then follows a print out of the initial values of the transform matrices and some key vectors. Then follows a header and then the real output records. The header is repeated here:

```
  time      x      y       z aspd xdot ydot zdot  psi thet  phi  lift  drag
```

The first column is time in seconds. As you can see, it prints only every hundredth time step. The next three columns are x, y, and z in feet. These coordinates are inertial (earth). A right hand rule is used with x and y in the plane of the earth's surface and z positive down. Thus, altitude is exactly the negative of z. This airplane is in a right hand sixty degree of bank level (nearly) turn. The fifth column is the airspeed in feet per second. It is calculated in the stability (body) axis.

The next three numbers (columns 6-8) are labeled xdot, ydot, and zdot, and are the time derivatives of the inertial position, expressed in inertial coordinates in feet per second. Their vector sum should equal the airspeed.

The next three numbers (columns 9-11) are the Euler angles psi (heading), thet (pitch), and phi (bank or roll) in degrees.

The last two numbers (columns 12-13) are the lift and drag, calculated in the stability axis, in pounds.

APPENDIX D

```
cat arcd.bat ast fs.c ast mv.c ast ..\vaxcfs\fs.c ast ..\vaxcfs\mv.c >tmp

*************************************************************************

/* Program fs,     3 degree of freedom     Flight Simulator            */
/* Frank J. Seiler Research Lab, USAF Academy, CO  ph 719-472-3122     */
/* Lt Col C Bruce Harmon and Maj Bill Dieterich, USAF Reserve, 1991    */
/* Final testing on Turbo C++ on Unisys Desktop III, MSDOS 4.01        */
/*   Also available under VAX/VMS 5.41                                 */
/*   Easily ported to Pascal, Fortran, Ada, and C                      */
/*   Easily ported to Unix and VMS                                     */
/* Documentation on line, same directory                               */
/* Test suite on line ..\fstest                                        */
/* last update (harmon)       8/29/91 Turbo C++ Unisys Desktop III     */
#include <stdio.h>
#include <math.h>
#define DTR   57.29577951
#define EE     2.71828183
#define PI     3.14159265
#define TWOPI  6.28318531
/* --------- externals -------------- */
extern mvmult(), mmmult();
extern double sin(), cos(), tan(), atan(), sqrt();
/* --------- globals ---------------- */
double  apsi[3][3],  athet[3][3],  aphi[3][3];
double aipsi[3][3], aithet[3][3], aiphi[3][3];
double   wt1[3],   wt2[3],   wt3[3],   wts[3];
double udot1[3], udot2[3], udot3[3], udots[3];
double xdot1[3], xdot2[3], xdot3[3], xdots[3],   x1[3];
double psi,thet,phi,psib,thetb,phib,psidot,thetdot,phidot,omegae;
double p,q,r,omegas,  t,tmin,tmax,dt,  thrust,lift,drag,weight;
double alt,aspd,rho,qbar,  mass,swing,cd0,aspect,eff;
double tclmin,tclmax,tclary[1000],clary[1000],dtcl;
int nptscl,imain,iprint,idebug,iconas;
/* ---------  main   ---------------- */
main()
{
        double temp, getcl();
        int i;
/* get simulation parameters */
        scanf("%lf %lf %lf %d %d %d",&tmin,&tmax,&dt,
          &iprint,&idebug,&iconas);
        printf("tmin,tmax,dt,iprint,idebug,iconas=");
        printf("%10.4f%10.4f%10.4f%4d%4d%4d\n",tmin,tmax,dt,
          iprint,idebug,iconas);
/* get aerodynamics parameters */
  scanf("%lf %lf %lf",&cd0,&aspect,&eff);
  printf("cd0,aspect,eff=");
  printf("%8.4f %8.4f %8.4f\n",cd0,aspect,eff);
  scanf("%lf %lf %lf", &swing,&weight,&thrust);
  printf("swing,weight,thrust=");
  printf("%12.0f %12.0f %16.4f\n",swing,weight,thrust);
        mass = weight / 32.174;
/* get initial conditions */
  scanf("%lf %lf %lf %lf %lf",&alt,&aspd,&psi,&thet,&phi);
  printf("alt,aspd,psi,thet,phi=");
  printf("%12.0f %8.0f %8.4f %8.4f %8.4f\n",alt,aspd,psi,thet,phi);
        rho = 0.0023769 * pow(EE,(-alt/30500.0));
        qbar = 0.5 * rho * aspd * aspd;
        printf("mass,rho,qbar=%10.2f %14.8f %12.4f\n",mass,rho,qbar);
```

```
/* get cl vs time curve */
        scanf("%lf %lf %d",&tclmin,&dtcl,&nptscl);
        tclmax = tclmin + dtcl * (nptscl - 1);
        printf("%8.2f %8.2f %6d\n",tclmin,dtcl,nptscl);
        for (i=0; i<nptscl; ++i) {
            tclary[i] = tclmin + dtcl * i;
            scanf("%lf",&temp);
            clary[i] = temp;
            printf("%8.2f %10.6f\n",tclary[i],clary[i]);
        }
/* initializations */
        t = tmin;
        temp = getcl();
        lift  = qbar * swing * temp;
        drag  = qbar * swing * (cd0 + temp*temp/(PI*aspect*eff));
        p = q = r = psidot = thetdot = phidot = 0.0;
        psib  = psi;
        thetb = thet;
        phib  = phi;
        wt1[0] = 0.0;
        wt1[1] = 0.0;
        wt1[2] = weight;
        x1[0] = 0.0;
        x1[1] = 0.0;
        x1[2] = -alt;
        xdots[0] = aspd;
        xdots[1] = 0.0;
        xdots[2] = 0.0;
/* get the transform matrices, based on Euler angles */
        get_mats(psi,thet,phi,apsi,athet,aphi);
        minv(apsi,aipsi);
        minv(athet,aithet);
        minv(aphi,aiphi);
/* convert stability xdots vector to inertial xdot1 */
        mvmult(aphi, xdots,xdot3);
        mvmult(athet,xdot3,xdot2);
        mvmult(apsi, xdot2,xdot1);
/* convert weight vector wt1 to stability wts     */
        mvmult(aipsi, wt1,wt2);
        mvmult(aithet,wt2,wt3);
        mvmult(aiphi, wt3,wts);
/* print initial vectors and matrices              */
        printf("apsi,athet,aphi\n");
        print_mat(apsi);
        print_mat(athet);
        print_mat(aphi);
        printf("aipsi,aithet,aiphi\n");
        print_mat(aipsi);
        print_mat(aithet);
        print_mat(aiphi);
        printf("wt1,wts,xdots,xdot1\n");
        print_vec(wt1);
        print_vec(wts);
        print_vec(xdots);
        print_vec(xdot1);
/* ----------- MAIN LOOP ---------------------- */
        imain = 0;
        printf("Output from program fs, flight sim for unsteady\n");
        printf("\n   time      x      y       z aspd xdot ydot ");
        printf("zdot  psi thet  phi   lift  drag\n\n");
```

```
while (t <= tmax) {
/* print output record if imain=0; imain ranges from
        zero to iprint-1, repeats */
        if (imain == 0) {
           printf("%7.2f",t);
           printf("%7.0f%7.0f%8.0f%5.0f",x1[0],x1[1],x1[2],aspd);
           printf("%5.0f%5.0f%5.0f",xdot1[0],xdot1[1],xdot1[2]);
           printf("%5.0f%5.0f%5.0f",psi*DTR,thet*DTR,phi*DTR);
           printf("%7.0f%6.0f\n",lift,drag);
        }
/* stop if below sea level */
        if (alt < 0.0) return;
/* find the forces acting on the body */
        temp = getcl();
        lift  = qbar * swing * temp;
        drag  = qbar * swing * (cd0 + temp*temp/(PI*aspect*eff));
/* get the transform matrices, based on Euler angles */
        get_mats(psi,thet,phi,apsi,athet,aphi);
        minv(apsi,aipsi);
        minv(athet,aithet);
        minv(aphi,aiphi);
/* convert the inertial weight vector wt1 to stability wts */
        mvmult(aipsi, wt1,wt2);
        mvmult(aithet,wt2,wt3);
        mvmult(aiphi, wt3,wts);
/* using eq 2-19, compute linear accel (f=ma), stab axes */
        udots[0] = (wts[0] - drag + thrust) / mass;
        udots[1] =  wts[1] / mass;
        udots[2] = (wts[2] - lift) / mass;
/* if constant airspeed profile, then set tangential accel to zero */
        if (iconas) udots[0] = 0.0;
/* airspeed change is entirely due to udots[0]              */
        aspd = aspd + dt * udots[0];
/* convert stability udots vector to inertial udot1 */
        mvmult(aphi, udots,udot3);
        mvmult(athet,udot3,udot2);
        mvmult(apsi, udot2,udot1);
        if (idebug) {
           printf("udots=  %8.2f%8.2f%8.2f\n",udots[0],udots[1],udots[2]);
           printf("udot1=  %8.2f%8.2f%8.2f\n",udot1[0],udot1[1],udot1[2]);
        }
/* advance xdot1 then x1 */
        for (i=0; i<3; i++) {
            xdot1[i] = xdot1[i] + dt * udot1[i];
            x1[i]    = x1[i]    + dt * xdot1[i];
        }
/* geometry to get psi, thet (phi constrained to its initial value) */
        psi = atan( xdot1[1] / xdot1[0] );
        if ( xdot1[0] < 0.0 )  psi = psi + PI;
          else if ( xdot1[1] < 0.0 ) psi = psi + TWOPI;
        psidot  = (psi  - psib)  / dt;
        if ( (psi-psib) > PI ) psidot = 0.0;
        if ( (psib-psi) > PI ) psidot = 0.0;

        temp = sqrt(xdot1[0]*xdot1[0] + xdot1[1]*xdot1[1]);
        thet = atan( -xdot1[2] / temp );
        thetdot = (thet - thetb) / dt;
        phidot  = 0.0;    /* pilot constraint */
/* back values for psi, thet, and phi        */
        psib = psi;
        thetb = thet;
        phib = phi;
/* use eq 2-46 to get p,q,r from psidot, thetdot, phidot    */
        p =  phidot - psidot * sin (thet);
        q =  thetdot * cos(phi) + psidot * cos(thet) * sin(phi);
        r = -thetdot * sin(phi) + psidot * cos(thet) * cos(phi);
        omegas = sqrt(p*p + q*q + r*r);  /* magnitude */
        omegae = sqrt(psidot*psidot + thetdot*thetdot + phidot*phidot);
        if (idebug) {
           printf("Eudots= %8.4f%8.4f%8.4f%8.4f\n",psidot*DTR,
             thetdot*DTR,phidot*DTR,omegae*DTR);
           printf("p,q,r=  %8.4f%8.4f%8.4f%8.4f\n",p*DTR,q*DTR,r*DTR,
             omegas*DTR);
        }
/* update the rest */
        alt = -x1[2];
        rho = 0.0023769 * pow( E,(-alt/30500.0));
        qbar = 0.5 * rho * aspd * aspd;
t = t + dt;
imain = imain + 1;
if (imain >= iprint) imain = 0;  /* print control */
}
}
/* --------------- FUNCTIONS -------------------- */
double getcl()
{
        double frac;
        int i;
        if (t < tclmin || t > tclmax) {
           printf("Error in getcl, t,tclmin,tclmax = ");
           printf("%10.3f %10.3f %10.3f\n",t,tclmin,tclmax);
           return 1.0;
        }
        i=0;
        while (i < nptscl && t >= tclary[i]) i++;
        frac = (t - tclary[i-1]) / dtcl;
        return (clary[i-1] + frac * (clary[i] - clary[i-1]));
}

*************************************************************************

/* file mv.c. matrix and vector functions, cb harmon, 3/91 */
/* last mod 8/10/91 Turbo C++ on Unisys Desktop III        */
#include <stdio.h>
#include <math.h>
extern double sin(), cos();

/* -------------------------------------------------------- */
print_mat(mat)
        double mat[][3];
{
        int i,j;
        for (i=0; i<3; ++i) {
            for (j=0; j<3; ++j)
                printf("%10.2f",mat[i][j]);
            printf("\n");
        }
        printf("\n");
}
/* -------------------------------------------------------- */
```

```
print_vec(vec)
        double vec[];
{
        int i;
        for (i=0; i<3; ++i)
            printf("%10.2f\n",vec[i]);
        printf("\n");
}
/* -------------------------------------------------------- */
mvmult(mata,vecb,vecc)
        double mata[][3], vecb[], vecc[];
{
        int i;
        for (i=0; i<3; ++i)
        vecc[i] = mata[i][0] * vecb[0]
                + mata[i][1] * vecb[1]
                + mata[i][2] * vecb[2];
}
/* -------------------------------------------------------- */
mmmult(mata,matb,matc)
        double mata[][3], matb[][3], matc[][3];
{
        int i,j;
        for (i=0; i<3; ++i)
        for (j=0; j<3; ++j)
        matc[i][j] = mata[i][0] * matb[0][j]
                   + mata[i][1] * matb[1][j]
                   + mata[i][2] * matb[2][j];
}
/* -------------------------------------------------------- */
double det(mat)
        double mat[][3];
{
        double d;
        d= mat[0][0] * (mat[1][1] * mat[2][2]
                     -  mat[2][1] * mat[1][2])
         - mat[0][1] * (mat[1][0] * mat[2][2]
                     -  mat[2][0] * mat[1][2])
         + mat[0][2] * (mat[1][0] * mat[2][1]
                     -  mat[2][0] * mat[1][1]);
        return d;
}
/* -------------------------------------------------------- */
minv(mata,matb)
        double mata[][3], matb[][3];
{
        double d, det();
        d = det(mata);
        if (d == 0.0) {
            printf("Div by zero in minv\n");
            return;
        }
        else {
        matb[0][0] =    (mata[1][1] * mata[2][2]
                       - mata[2][1] * mata[1][2])/d;
        matb[0][1] =  - (mata[0][1] * mata[2][2]
                       - mata[2][1] * mata[0][2])/d;
        matb[0][2] =    (mata[0][1] * mata[1][2]
                       - mata[1][1] * mata[0][2])/d;
        matb[1][0] =  - (mata[1][0] * mata[2][2]
                      - mata[2][0] * mata[1][2])/d;
        matb[1][1] =    (mata[0][0] * mata[2][2]
                      - mata[2][0] * mata[0][2])/d;
        matb[1][2] =  - (mata[0][0] * mata[1][2]
                      - mata[1][0] * mata[0][2])/d;
        matb[2][0] =    (mata[1][0] * mata[2][1]
                      - mata[2][0] * mata[1][1])/d;
        matb[2][1] =  - (mata[0][0] * mata[2][1]
                      - mata[2][0] * mata[0][1])/d;
        matb[2][2] =    (mata[0][0] * mata[1][1]
                      - mata[1][0] * mata[0][1])/d;
        }
}
/* ------------------------------------------------------------ */
/* gets the rotation matrices for inertial to stability axes */
/* inputs are psi, thet, and phi in radians                   */
/* see derivation in roskam chap 2, p27                       */
get_mats(psi, thet, phi, mat_psi, mat_thet, mat_phi)
        double psi, thet, phi;
        double mat_psi[][3], mat_thet[][3], mat_phi[][3];
{
        mat_psi[0][0]  =  cos(psi);
        mat_psi[0][1]  = -sin(psi);
        mat_psi[0][2]  =  0.0;
        mat_psi[1][0]  =  sin(psi);
        mat_psi[1][1]  =  cos(psi);
        mat_psi[1][2]  =  0.0;
        mat_psi[2][0]  =  0.0;
        mat_psi[2][1]  =  0.0;
        mat_psi[2][2]  =  1.0;

        mat_thet[0][0] =  cos(thet);
        mat_thet[0][1] =  0.0;
        mat_thet[0][2] =  sin(thet);
        mat_thet[1][0] =  0.0;
        mat_thet[1][1] =  1.0;
        mat_thet[1][2] =  0.0;
        mat_thet[2][0] = -sin(thet);
        mat_thet[2][1] =  0.0;
        mat_thet[2][2] =  cos(thet);

        mat_phi[0][0]  =  1.0;
        mat_phi[0][1]  =  0.0;
        mat_phi[0][2]  =  0.0;
        mat_phi[1][0]  =  0.0;
        mat_phi[1][1]  =  cos(phi);
        mat_phi[1][2]  = -sin(phi);
        mat_phi[2][0]  =  0.0;
        mat_phi[2][1]  =  sin(phi);
        mat_phi[2][2]  =  cos(phi);
}

*********************************************************************

/* Program fs,     3 degree of freedom     Flight Simulator          */
/* Frank J. Seiler Research Lab, USAF Academy, CO  ph 719-472-3122   */
/* Lt Col C Bruce Harmon and Maj Bill Dieterich, USAF Reserve, 1991  */
/* Final testing on Turbo C++ on Unisys Desktop III, MSDOS 4.01      */
/*   Also available under VAX/VMS 5.41                                */
/*   Easily ported to Pascal, Fortran, Ada, and C                     */
/*   Easily ported to Unix and VMS                                     */
/* Documentation on line, same directory                               */
/* Test suite on line ..\fstest                                        */
/* last update (harmon)       8/29/91        VAX/VMS 5.41              */
#include stdio
#include math
#define DTR   57.29577951
#define EE     2.71828183
#define PI     3.14159265
#define TWOPI  6.28318531
/* --------- externals -------------- */
extern mvmult(), mmmult();
extern double sin(), cos(), tan(), atan(), sqrt();
/* --------- globals ---------------- */
double  apsi[3][3],  athet[3][3],  aphi[3][3];
double aipsi[3][3], aithet[3][3], aiphi[3][3];
double   wt1[3],   wt2[3],   wt3[3],   wts[3];
double udot1[3], udot2[3], udot3[3], udots[3];
double xdot1[3], xdot2[3], xdot3[3], xdots[3],   x1[3];
double psi,thet,phi,psib,thetb,phib,psidot,thetdot,phidot,omegae;
double p,q,r,omegas,  t,tmin,tmax,dt,  thrust,lift,drag,weight;
double alt,aspd,rho,qbar,  mass,swing,cd0,aspect,eff;
double tclmin,tclmax,tclary[1000],clary[1000],dtcl;
int npts cl,imain,iprint,idebug,iconas;
/* ---------  main   ---------------- */
main()
{
        double temp, getcl();
        int i;
/* get simulation parameters */
        scanf("%lf %lf %lf %d %d %d",&tmin,&tmax,&dt,
          &iprint,&idebug,&iconas);
        printf("tmin,tmax,dt,iprint,idebug,iconas=");
        printf("%10.4f%10.4f%10.4f%4d%4d%4d\n",tmin,tmax,dt,
          iprint,idebug,iconas);
/* get aerodynamics parameters */
  scanf("%lf %lf %lf",&cd0,&aspect,&eff);
  printf("cd0,aspect,eff=");
  printf("%8.4f %8.4f %8.4f\n",cd0,aspect,eff);
  scanf("%lf %lf %lf", &swing,&weight,&thrust);
  printf("swing,weight,thrust=");
  printf("%12.0f %12.0f %16.4f\n",swing,weight,thrust);
        mass = weight / 32.174;
/* get initial conditions */
  scanf("%lf %lf %lf %lf %lf",&alt,&aspd,&psi,&thet,&phi);
  printf("alt,aspd,psi,thet,phi=");
  printf("%12.0f %8.0f %8.4f %8.4f %8.4f\n",alt,aspd,psi,thet,phi);
        rho = 0.0023769 * pow(EE,(-alt/30500.0));
        qbar = 0.5 * rho * aspd * aspd;
        printf("mass,rho,qbar=%10.2f %14.8f %12.4f\n",mass,rho,qbar);
/* get cl vs time curve */
        scanf("%lf %lf %d",&tclmin,&dtcl,&nptscl);
        tclmax = tclmin + dtcl * (nptscl - 1);
        printf("%8.2f %8.2f %6d\n",tclmin,dtcl,nptscl);
        for (i=0; i<nptscl; ++i) {
            tclary[i] = tclmin + dtcl * i;
            scanf("%lf",&temp);
            clary[i] = temp;
            printf("%8.2f %10.6f\n",tclary[i],clary[i]);
        }
```

```
/* initializations */
       t = tmin;
       temp = getcl();
       lift  = qbar * swing * temp;
       drag  = qbar * swing * (cd0 + temp*temp/(PI*aspect*eff));
       p = q = r = psidot = thetdot = phidot = 0.0;
       psib  = psi;
       thetb = thet;
       phib  = phi;
       wt1[0] = 0.0;
       wt1[1] = 0.0;
       wt1[2] = weight;
       x1[0] = 0.0;
       x1[1] = 0.0;
       x1[2] = -alt;
       xdots[0] = aspd;
       xdots[1] = 0.0;
       xdots[2] = 0.0;
/* get the transform matrices, based on Euler angles */
       get_mats(psi,thet,phi,apsi,athet,aphi);
       minv(apsi,aipsi);
       minv(athet,aithet);
       minv(aphi,aiphi);
/* convert stability xdots vector to inertial xdot1 */
       mvmult(aphi, xdots,xdot3);
       mvmult(athet,xdot3,xdot2);
       mvmult(apsi, xdot2,xdot1);
/* convert weight vector wt1 to stability wts     */
       mvmult(aipsi, wt1,wt2);
       mvmult(aithet,wt2,wt3);
       mvmult(aiphi, wt3,wts);
/* print initial vectors and matrices             */
       printf("apsi,athet,aphi\n");
       print_mat(apsi);
       print_mat(athet);
       print_mat(aphi);
       printf("aipsi,aithet,aiphi\n");
       print_mat(aipsi);
       print_mat(aithet);
       print_mat(aiphi);
       printf("wt1,wts,xdots,xdot1\n");
       print_vec(wt1);
       print_vec(wts);
       print_vec(xdots);
       print_vec(xdot1);
/* ----------- MAIN LOOP ---------------------- */
       imain = 0;
       printf("Output from program fs, flight sim for unsteady\n");
       printf("\n   time      x      y       z aspd xdot ydot ");
       printf("zdot  psi thet  phi   lift  drag\n\n");
while (t <= tmax) {
/* print output record if imain=0; imain ranges from
        zero to iprint-1, repeats */
       if (imain == 0) {
          printf("%7.2f",t);
          printf("%7.0f%7.0f%8.0f%5.0f",x1[0],x1[1],x1[2],aspd);
          printf("%5.0f%5.0f%5.0f",xdot1[0],xdot1[1],xdot1[2]);
          printf("%5.0f%5.0f%5.0f",psi*DTR,thet*DTR,phi*DTR);
          printf("%7.0f%6.0f\n",lift,drag);
       }
```

```
/* stop if below sea level */
       if (alt < 0.0) return;
/* find the forces acting on the body */
       temp = getcl();
       lift  = qbar * swing * temp;
       drag  = qbar * swing * (cd0 + temp*temp/(PI*aspect*eff));
/* get the transform matrices, based on Euler angles */
       get_mats(psi,thet,phi,apsi,athet,aphi);
       minv(apsi,aipsi);
       minv(athet,aithet);
       minv(aphi,aiphi);
/* convert the inertial weight vector wt1 to stability wts */
       mvmult(aipsi, wt1,wt2);
       mvmult(aithet,wt2,wt3);
       mvmult(aiphi, wt3,wts);
/* using eq 2-19, compute linear accel (f=ma), stab axes */
       udots[0] = (wts[0] - drag + thrust) / mass;
       udots[1] =  wts[1] / mass;
       udots[2] = (wts[2] - lift) / mass;
/* if constant airspeed profile, then set tangential accel to zero */
       if (iconas) udots[0] = 0.0;
/* airspeed change is entirely due to udots[0]          */
       aspd = aspd + dt * udots[0];
/* convert stability udots vector to inertial udot1 */
       mvmult(aphi, udots,udot3);
       mvmult(athet,udot3,udot2);
       mvmult(apsi, udot2,udot1);
       if (idebug) {
          printf("udots=  %8.2f%8.2f%8.2f\n",udots[0],udots[1],udots[2]);
          printf("udot1=  %8.2f%8.2f%8.2f\n",udot1[0],udot1[1],udot1[2]);
       }
/* advance xdot1 then x1 */
       for (i=0; i<3; i++) {
           xdot1[i] = xdot1[i] + dt * udot1[i];
           x1[i]    = x1[i]    + dt * xdot1[i];
       }
/* geometry to get psi, thet (phi constrained to its initial value) */
       psi = atan( xdot1[1] / xdot1[0] );
       if ( xdot1[0] < 0.0 )  psi = psi + PI;
         else if ( xdot1[1] < 0.0 ) psi = psi + TWOPI;
       psidot  = (psi  - psib)  / dt;
       if ( (psi-psib) > PI ) psidot = 0.0;
       if ( (psib-psi) > PI ) psidot = 0.0;

       temp = sqrt(xdot1[0]*xdot1[0] + xdot1[1]*xdot1[1]);
       thet = atan( -xdot1[2] / temp );
       thetdot = (thet - thetb) / dt;
       phidot  = 0.0;    /* pilot constraint */
/* back values for psi, thet, and phi        */
       psib = psi;
       thetb = thet;
       phib = phi;
/* use eq 2-46 to get p,q,r from psidot, thetdot, phidot     */
       p =  phidot - psidot * sin (thet);
       q =  thetdot * cos(phi) + psidot * cos(thet) * sin(phi);
       r = -thetdot * sin(phi) + psidot * cos(thet) * cos(phi);
       omegas = sqrt(p*p + q*q + r*r);  /* magnitude */
       omegae = sqrt(psidot*psidot + thetdot*thetdot + phidot*phidot);
       if (idebug) {
          printf("Eudots= %8.4f%8.4f%8.4f%8.4f\n",psidot*DTR,
            thetdot*DTR,phidot*DTR,omegae*DTR);
         printf("p,q,r=  %8.4f%8.4f%8.4f%8.4f\n",p*DTR,q*DTR,r*DTR,
            omegas*DTR);
      }
/* update the rest */
      alt = -x1[2];
      rho = 0.0023769 * pow(EE,(-alt/30500.0));
      qbar = 0.5 * rho * aspd * aspd;
t = t + dt;
imain = imain + 1;
if (imain >= iprint) imain = 0;  /* print control */
}
}
/* -------------- FUNCTIONS ------------------- */
double getcl()
{
      double frac;
      int i;
      if (t < tclmin || t > tclmax) {
         printf("Error in getcl, t,tclmin,tclmax = ");
         printf("%10.3f %10.3f %10.3f\n",t,tclmin,tclmax);
         return 1.0;
      }
      i=0;
      while (i < nptscl && t >= tclary[i]) i++;
      frac = (t - tclary[i-1]) / dtcl;
      return (clary[i-1] + frac * (clary[i] - clary[i-1]));
}

**********************************************************************

/* file mv.c, matrix and vector functions, cb harmon, 3/91 */
#include stdio
#include math
extern double sin(), cos();

/* ------------------------------------------------------- */
print_mat(mat)
      double mat[][3];
{
      int i,j;
      for (i=0; i<3; ++i) {
         for (j=0; j<3; ++j)
            printf("%10.2f",mat[i][j]).
         printf("\n");
      }
      printf("\n");
}
/* ------------------------------------------------------- */
print_vec(vec)
      double vec[];
{
      int i;
      for (i=0; i<3; ++i)
         printf("%10.2f\n",vec[i]);
      printf("\n");
}
/* ------------------------------------------------------- */
mvmult(mata,vecb,vecc)
      double mata[][3], vecb[], vecc[];
```

```
{
        int i;
        for (i=0; i<3; ++i)
        vecc[i] = mata[i][0] * vecb[0]
                + mata[i][1] * vecb[1]
                + mata[i][2] * vecb[2];
}
/* ---------------------   ------------------------------ */
mmmult(mata,matb,matc)
        double mata[][3], matb[][3], matc[][3];
{
        int i,j;
        for (i=0; i<3; ++i)
        for (j=0; j<3; ++j)
        matc[i][j] = mata[i][0] * matb[0][j]
                   + mata[i][1] * matb[1][j]
                   + mata[i][2] * matb[2][j];
}
/* ------------------------------------------------------- */
double det(mat)
        double mat[][3];
{
        double d;
        d= mat[0][0] * (mat[1][1] * mat[2][2]
                     -  mat[2][1] * mat[1][2])
         - mat[0][1] * (mat[1][0] * mat[2][2]
                     -  mat[2][0] * mat[1][2])
         + mat[0][2] * (mat[1][0] * mat[2][1]
                     -  mat[2][0] * mat[1][1]);
        return d;
}
/* ------------------------------------------------------- */
minv(mata,matb)
        double mata[][3], matb[][3];
{
        double d, det();
        d = det(mata);
        if (d == 0.0) {
            printf("Div by zero in minv\n");
            return;
        }
        else {
        matb[0][0] =    (mata[1][1] * mata[2][2]
                       - mata[2][1] * mata[1][2])/d;
        matb[0][1] =  - (mata[0][1] * mata[2][2]
                       - mata[2][1] * mata[0][2])/d;
        matb[0][2] =    (mata[0][1] * mata[1][2]
                       - mata[1][1] * mata[0][2])/d;
        matb[1][0] =  - (mata[1][0] * mata[2][2]
                       - mata[2][0] * mata[1][2])/d;
        matb[1][1] =    (mata[0][0] * mata[2][2]
                       - mata[2][0] * mata[0][2])/d;
        matb[1][2] =  - (mata[0][0] * mata[1][2]
                       - mata[1][0] * mata[0][2])/d;
        matb[2][0] =    (mata[1][0] * mata[2][1]
                       - mata[2][0] * mata[1][1])/d;
        matb[2][1] =  - (mata[0][0] * mata[2][1]
                       - mata[2][0] * mata[0][1])/d;
        matb[2][2] =    (mata[0][0] * mata[1][1]
                       - mata[1][0] * mata[0][1])/d;
        }
}
/* --------------------------------------------------------------- */
/* gets the rotation matrices for inertial to stability axes */
/* inputs are psi, thet, and phi in radians                         */
/* see derivation in roskam chap 2, p27                              */
get_mats(psi, thet, phi, mat_psi, mat_thet, mat_phi)
        double psi, thet, phi;
        double mat_psi[][3], mat_thet[][3], mat_phi[][3];
{
        mat_psi[0][0]  =  cos(psi);
        mat_psi[0][1]  = -sin(psi);
        mat_psi[0][2]  =  0.0;
        mat_psi[1][0]  =  sin(psi);
        mat_psi[1][1]  =  cos(psi);
        mat_psi[1][2]  =  0.0;
        mat_psi[2][0]  =  0.0;
        mat_psi[2][1]  =  0.0;
        mat_psi[2][2]  =  1.0;

        mat_thet[0][0] =  cos(thet);
        mat_thet[0][1] =  0.0;
        mat_thet[0][2] =  sin(thet);
        mat_thet[1][0] =  0.0;
        mat_thet[1][1] =  1.0;
        mat_thet[1][2] =  0.0;
        mat_thet[2][0] = -sin(thet);
        mat_thet[2][1] =  0.0;
        mat_thet[2][2] =  cos(thet);

        mat_phi[0][0]  =  1.0;
        mat_phi[0][1]  =  0.0;
        mat_phi[0][2]  =  0.0;
        mat_phi[1][0]  =  0.0;
        mat_phi[1][1]  =  cos(phi);
        mat_phi[1][2]  = -sin(phi);
        mat_phi[2][0]  =  0.0;
        mat_phi[2][1]  =  sin(phi);
        mat_phi[2][2]  =  cos(phi);
}
```

APPENDIX E

```
**********************************************************************
*       program simdisp                                              *
**********************************************************************
*                                                                    *
*       Last Modified:  7 Apr 91   Maj Dieterich                     *
*                                                                    *
*       Description:                                                 *
*                                                                    *
*       This program plots 3-D curves representing the flight path   *
*       of a maneuvering aircraft. The program reads an input file   *
*       that contains the x,y and z components of the aircraft's     *
*       position as a f(t). The program is written in FORTRAN and    *
*       utilizes calls to the CA-DISSPLA graphics software package.  *
*       The output is a plot that can be displayed on either the     *
*       TEK 4107 terminal or the HP 7550 plotter.                    *
*       See the CA-DISSPLA documentation for making changes to the   *
*       program.                                                     *
**********************************************************************
*                                                                    *
*       Input:                                                       *
*                                                                    *
*       An input file named simdisp.dat is read in. The file format  *
*       is:                                                          *
*                                                                    *
*       npts                              ;number of points (integer) *
*       t(0)    x(0)    y(0)    z(0)      ;arrays of time,x,y,z (reals)*
*       t(1)...                                                      *
*       .                                                            *
*       .                                                            *
*       .                                                            *
**********************************************************************
*                                                                    *
*       Compiling/Linking/Running:                                   *
*                                                                    *
*       To compile on the VAX:   for  simdisp.for                    *
*                                                                    *
*       To link on the VAX:      link simdisp,disshr11/l             *
*                                                                    *
*       To run on the VAX:       run simdisp                         *
*                                                                    *
*              for the TEK 4107: a picture is displayed              *
*                                                                    *
*              for the HP 7550 : a data file called std00001.dat     *
*                                is created.                         *
*                  to plot type: print /que=nhplot std00001.dat;     *
*                                                                    *
**********************************************************************
```

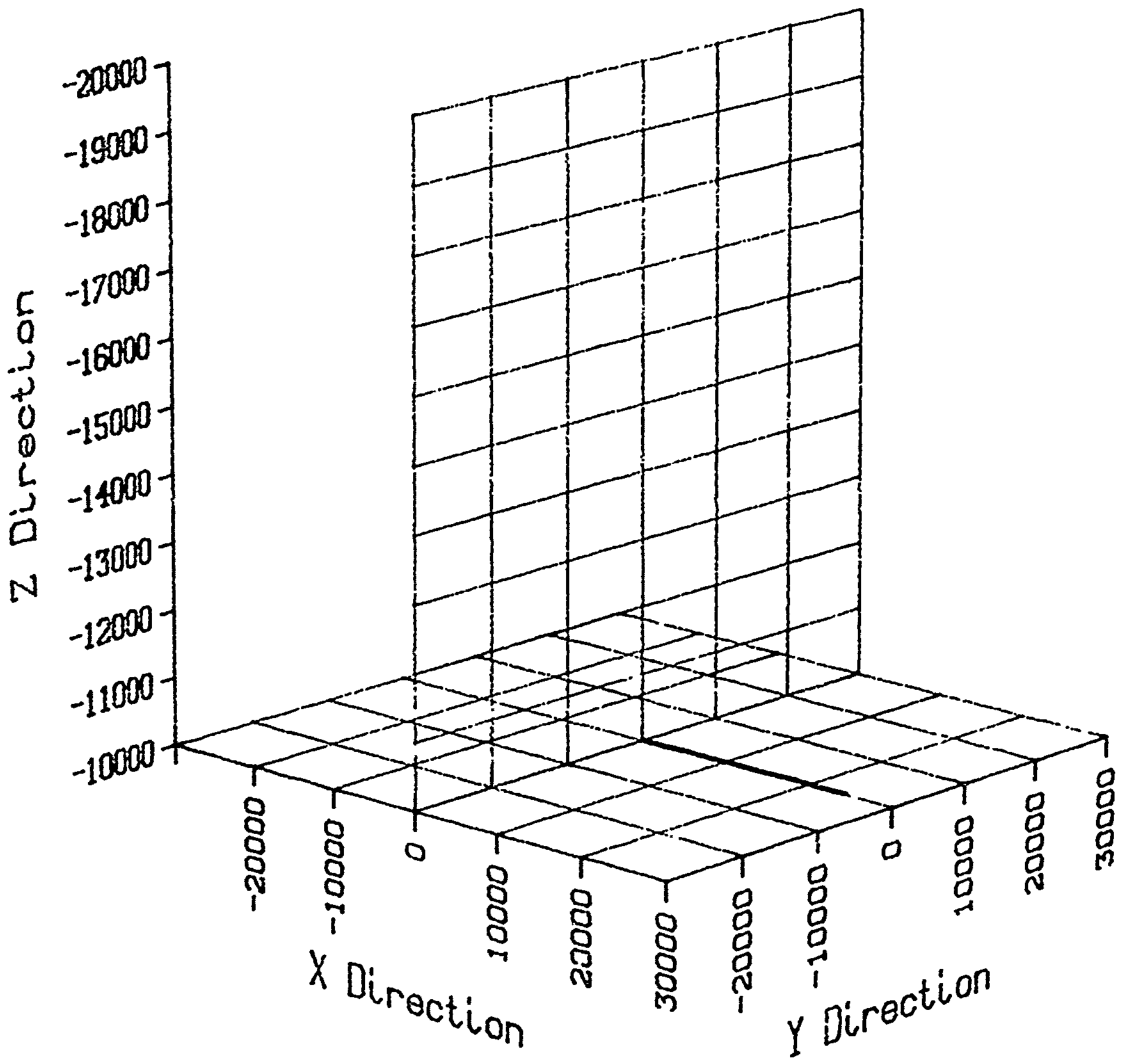
```
c variable declarations

        real t(500),x(500),y(500),z(500)
        integer npts,i,ndev,nvp,nmb
        integer ibuf(16)
        real xvu,yvu,zvu,size,half,yzplot



c  change view point
  3     type *,'do you want to change the viewpoint?
     +  (1=yes,0=no)'
        read(5,*) nvp

        if (nvp .lt. 0 .or. nvp .gt. 1) go to 3
        if (nvp .eq. 1) then
                type *,'enter x,y,z viewpoint
     +          e.g.     100.  -100    40.'
                read (5,*)xvu,yvu,zvu
        else
                xvu=100.
                yvu=-100.
                zvu=40.
        end if


c  change size of maneuver box
  4     type *,'do you want to change the size of the maneuver box?
     +  (1=yes,0=no)'
        read(5,*) nmb

        if (nmb .lt. 0 .or. nmb .gt. 1) go to 4
        if (nmb .eq. 1) then
                type *,'enter size of maneuver box
     +          e.g.     6000.'
                read (5,*)size
        else
                size=60000.

        end if
        half=size/2.

c  select graphics output device

    2   type *,'enter device num(0=tek4107,1=plotter):'
        read(5,*)ndev

        if (ndev .lt. 0 .or. ndev .gt. 1) go to 2
        if (ndev .eq. 0) call tk41do(1,4107)
        if (ndev .eq. 1) then
                call iomgr(ibuf,-1)
                ibuf(1)=5
                call iomgr(ibuf,-102)
                ibuf(1)=0
                call iomgr(ibuf,-104)
                call hp7550(1)
        end if
```

```
c  open input file
        call assign(7,'simdisp.dat')

c       read in the data from logical 7, simdisp.dat
        read(7,*)npts
        i=1
        do while (i .le. npts)
                read(7,*)t(i),x(i),y(i),z(i)
                i=i+1
        end do



c  set z axis numbering parallel to a y plane
        call zaxang(-90.)
        call xaxang(90.)
        call yaxang(-90.)


c  set up the title and plot area
        if (ndev .eq. 1) call hwrot('movie')
        call page(8.5,11.)
        call nobrdr
        call area2d(6.5,8.)
        call height(.20)
        call newclr('blue')
        call headin('AIRCRAFT FLIGHT PATH$',100,1.3,1)


c  set up axis labels and define 3d work box
        call x3name('X Direction$',100)
        call y3name('Y Direction$',100)
        call z3name('Z Direction$',100)
        call newclr('whit')
        call volm3d(10.,10.,10.)



c  establish view point in work box units
        call vuabs(xvu,yvu,zvu)


c  use integer numbering on axes
        call intaxs


c  adjust axes labels to avoid interference
        call xnmadj('ends')
        call ynmadj('ends')
        call znmadj('ends')


c  set scaling limits for work box data

        call graf3d(-half,10000.,half,-half,10000.,half,-5000.,
     +  1000.,-15000.)

c  set curve marker symbol and draw
c  raspln interpolated 3d curve
        call marker(15)
c       call raspln(0.)
        call newclr('red')
        call curv3d(x,y,z,npts,0)
        call newclr('whit')


c  the following code is optional. The lines drawn may in certain
c  cases improve the readability of the 3-d curve
c  draw vertical and horizontal lines from curve
c       to walls of box
c       do 100 i=1,npts
c               call rlvec3(x(i),y(i),0.0,x(i),y(i),z(i),0)
c               call rlvec3(x(i),y(i),z(i),-half,y(i),z(i),0)
c 100   continue

c  put 2d plot of  x y data on floor of 3d box
        call grfiti(0.,0.,5.,1.,0.,5.,0.,1.0,5.)
        call area2d(10.,10.)
        call graf(-half,10000.,half,-half,10000.,half)
        call newclr('blue')
        call marker (4)
        call curve(x,y,npts,0)
        call newclr('whit')
        call grid(1,1)
        call end3gr(-1)


c  put 2d plot of  y z on back wall or thru x=0 of 3d box
        call grfiti(5.,0.,0.,5.,1.,0.,5.,0.,1.)
        call area2d(10.,10.)
        call graf(-half,10000.,half,-5000.,1000.,-15000.)
        call newclr('blue')
        call marker (4)
c       call curve(y,z,npts,0)
        call newclr('whit')
        call grid(1,1)
        call end3gr(-2)

c  draw a box around plot
c       call box3d

c  terminate plot

        call endpl(0)
        call donepl
        stop
        end
```

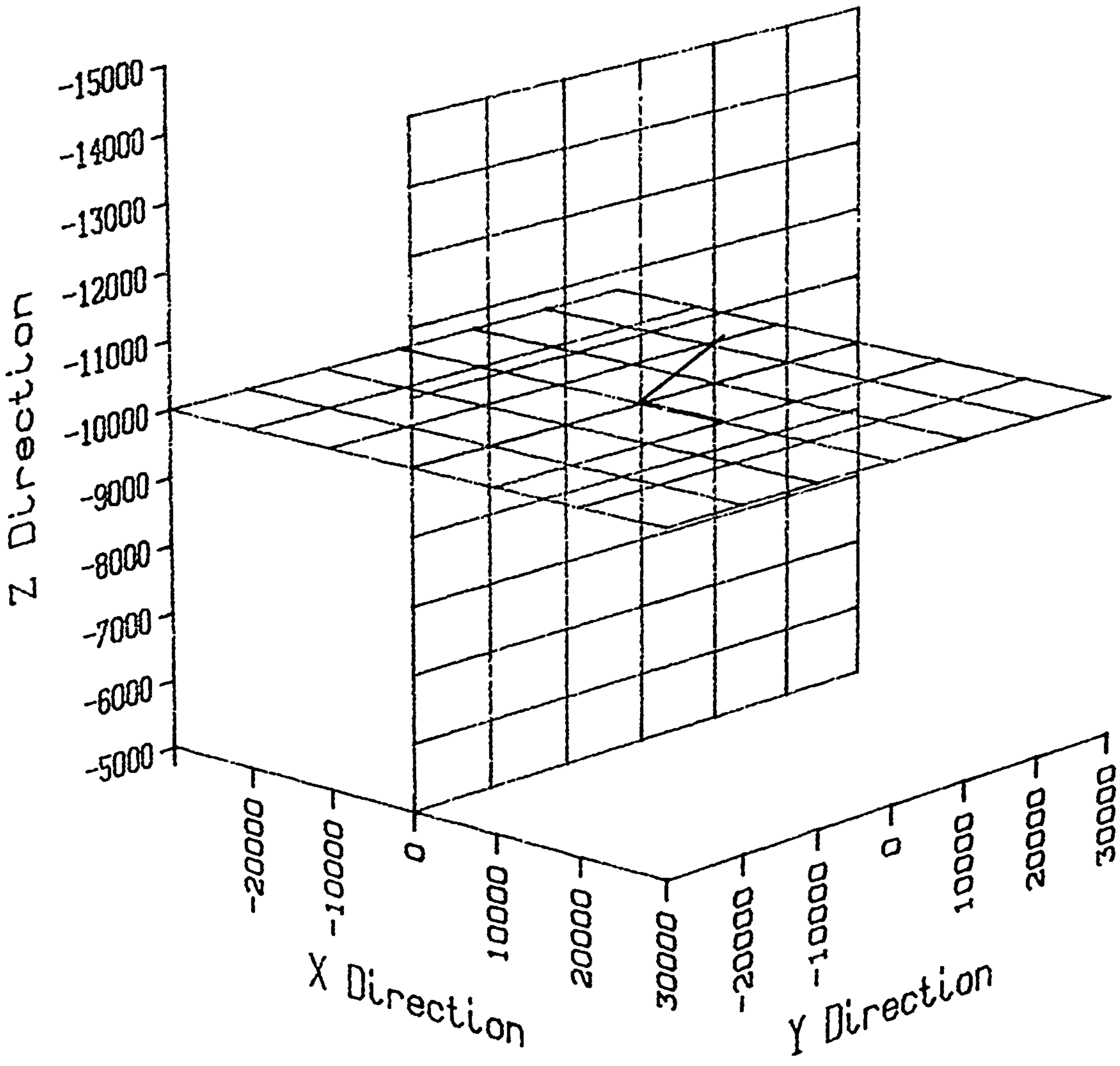
APPENDIX F

0.00	0	0	-10000	500	499	0	26	0	-3	60	71901	4941
1.00	498	28	-9974	500	496	56	26	6	-3	60	71903	4941
2.00	990	111	-9948	500	487	110	26	13	-3	60	71905	4940
3.00	1470	249	-9921	500	472	164	26	19	-3	60	71907	4939
4.00	1932	439	-9895	500	451	215	26	26	-3	60	71909	4938
5.00	2369	679	-9869	500	424	264	26	32	-3	60	71911	4937
6.00	2778	967	-9843	500	392	310	26	38	-3	60	71912	4937
7.00	3151	1298	-9817	500	355	351	26	45	-3	60	71914	4936
8.00	3486	1668	-9791	500	314	389	26	51	-3	60	71915	4935
9.00	3777	2074	-9765	500	269	421	26	57	-3	60	71917	4934
10.00	4022	2509	-9738	500	220	448	26	64	-3	60	71918	4934
11.00	4216	2969	-9712	500	169	470	26	70	-3	60	71920	4933
12.00	4358	3448	-9686	500	116	486	26	77	-3	60	71921	4932
13.00	4447	3939	-9660	500	61	496	26	83	-3	60	71922	4931
14.00	4480	4438	-9634	500	5	500	26	89	-3	60	71923	4930
15.00	4457	4937	-9608	500	-50	497	26	96	-3	60	71924	4930
16.00	4379	5430	-9582	500	-105	489	26	102	-3	60	71925	4929
17.00	4247	5912	-9556	500	-159	474	26	109	-3	60	71926	4928
18.00	4061	6376	-9530	500	-210	453	26	115	-3	60	71927	4927
19.00	3826	6817	-9504	500	-260	427	26	121	-3	60	71928	4926
20.00	3543	7228	-9478	500	-305	396	26	128	-3	60	71929	4926
21.00	3216	7606	-9452	500	-348	359	26	134	-3	60	71929	4925
22.00	2849	7945	-9426	500	-385	319	26	140	-3	60	71930	4924
23.00	2446	8242	-9400	500	-418	274	26	147	-3	60	71930	4923
24.00	2013	8491	-9374	500	-446	226	26	153	-3	60	71931	4922
25.00	1555	8691	-9348	500	-469	175	26	160	-3	60	71931	4922
26.00	1078	8839	-9322	500	-485	121	26	166	-3	60	71931	4921
27.00	587	8933	-9296	500	-496	67	26	172	-3	60	71931	4920
28.00	88	8972	-9270	500	-500	11	26	179	-3	60	71931	4919
29.00	-411	8955	-9244	500	-498	-44	26	185	-3	60	71931	4918
30.00	-906	8883	-9218	500	-490	-99	26	191	-3	60	71931	4917
31.00	-1390	8757	-9192	500	-476	-153	26	198	-3	60	71931	4917
32.00	-1856	8577	-9166	500	-456	-205	26	204	-3	60	71931	4916
33.00	-2300	8347	-9140	500	-431	-255	26	211	-3	60	71930	4915
34.00	-2716	8069	-9114	500	-400	-301	26	217	-3	60	71930	4914
35.00	-3098	7746	-9089	500	-364	-343	26	223	-3	60	71929	4913
36.00	-3442	7383	-9063	500	-324	-382	26	230	-3	60	71929	4912
37.00	-3744	6984	-9037	500	-279	-415	26	236	-3	60	71928	4912
38.00	-3999	6554	-9011	500	-232	-444	26	242	-3	60	71927	4911
39.00	-4205	6098	-8985	500	-181	-467	26	249	-3	60	71926	4910
40.00	-4360	5622	-8959	500	-128	-484	26	255	-3	60	71926	4909
41.00	-4460	5132	-8934	500	-73	-495	26	262	-3	60	71925	4908
42.00	-4506	4634	-8908	500	-18	-500	26	268	-3	60	71924	4907
43.00	-4496	4133	-8882	500	38	-499	26	274	-3	60	71922	4906
44.00	-4430	3637	-8856	500	93	-492	26	281	-3	60	71921	4906
45.00	-4310	3151	-8830	500	147	-479	26	287	-3	60	71920	4905
46.00	-4137	2682	-8805	500	199	-460	26	293	-3	60	71919	4904
47.00	-3912	2234	-8779	500	249	-435	26	300	-3	60	71917	4903
48.00	-3640	1815	-8753	500	295	-404	26	306	-3	60	71916	4902
49.00	-3322	1428	-8728	500	338	-369	26	313	-3	60	71914	4901
50.00	-2964	1078	-8702	500	377	-329	26	319	-3	60	71913	4900

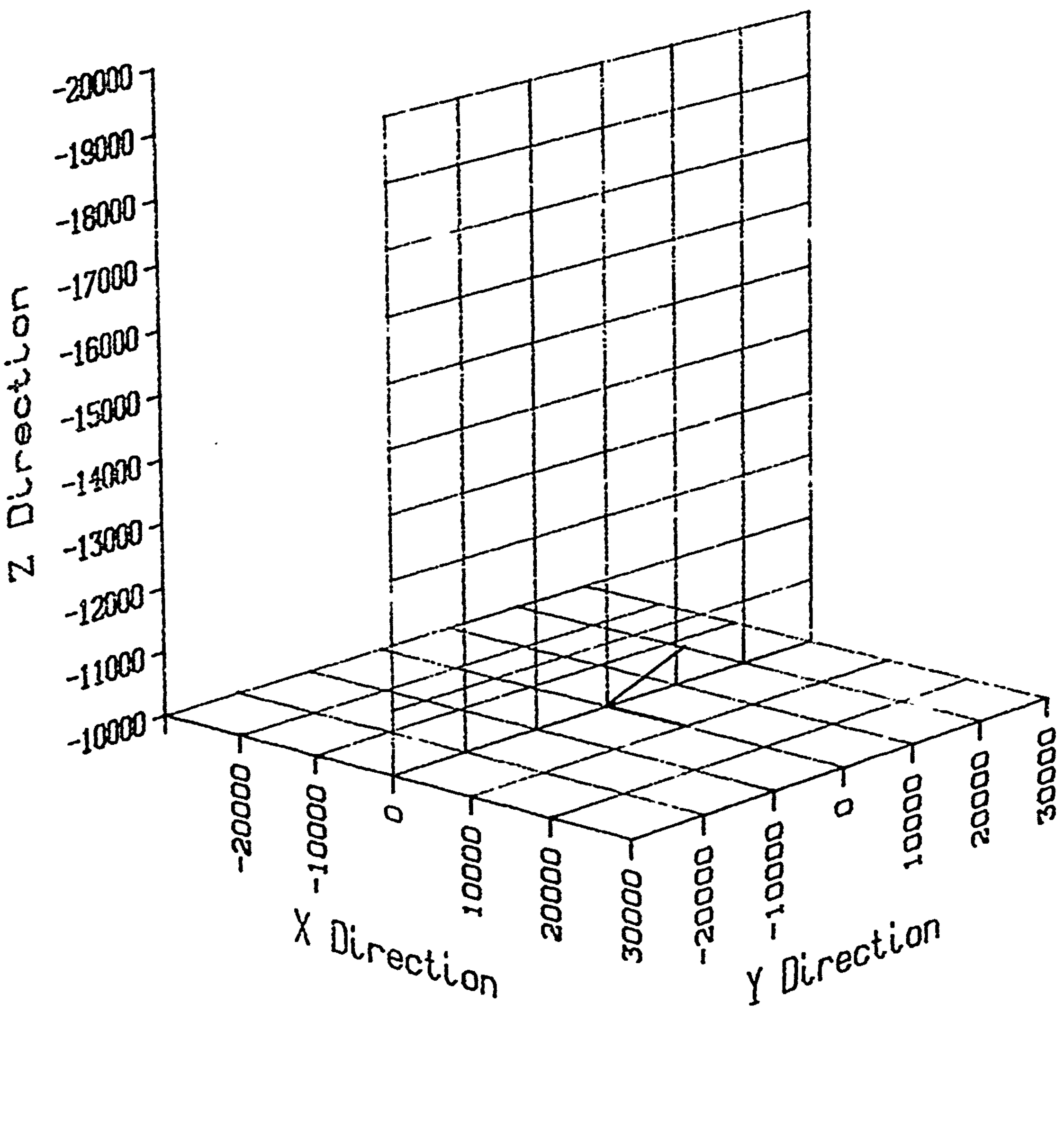
AIRCRAFT FLIGHT PATH



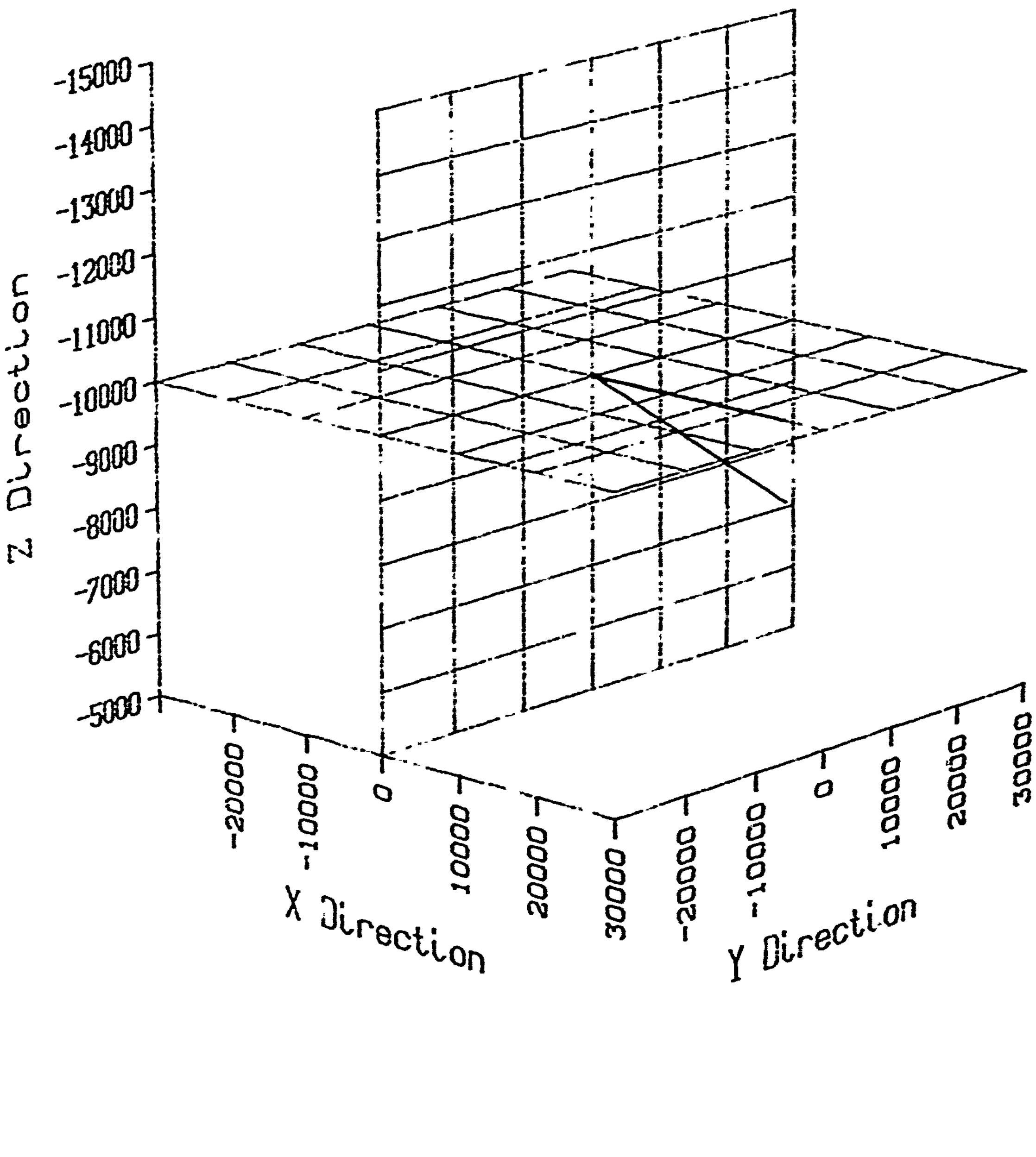
AIRCRAFT FLIGHT PATH



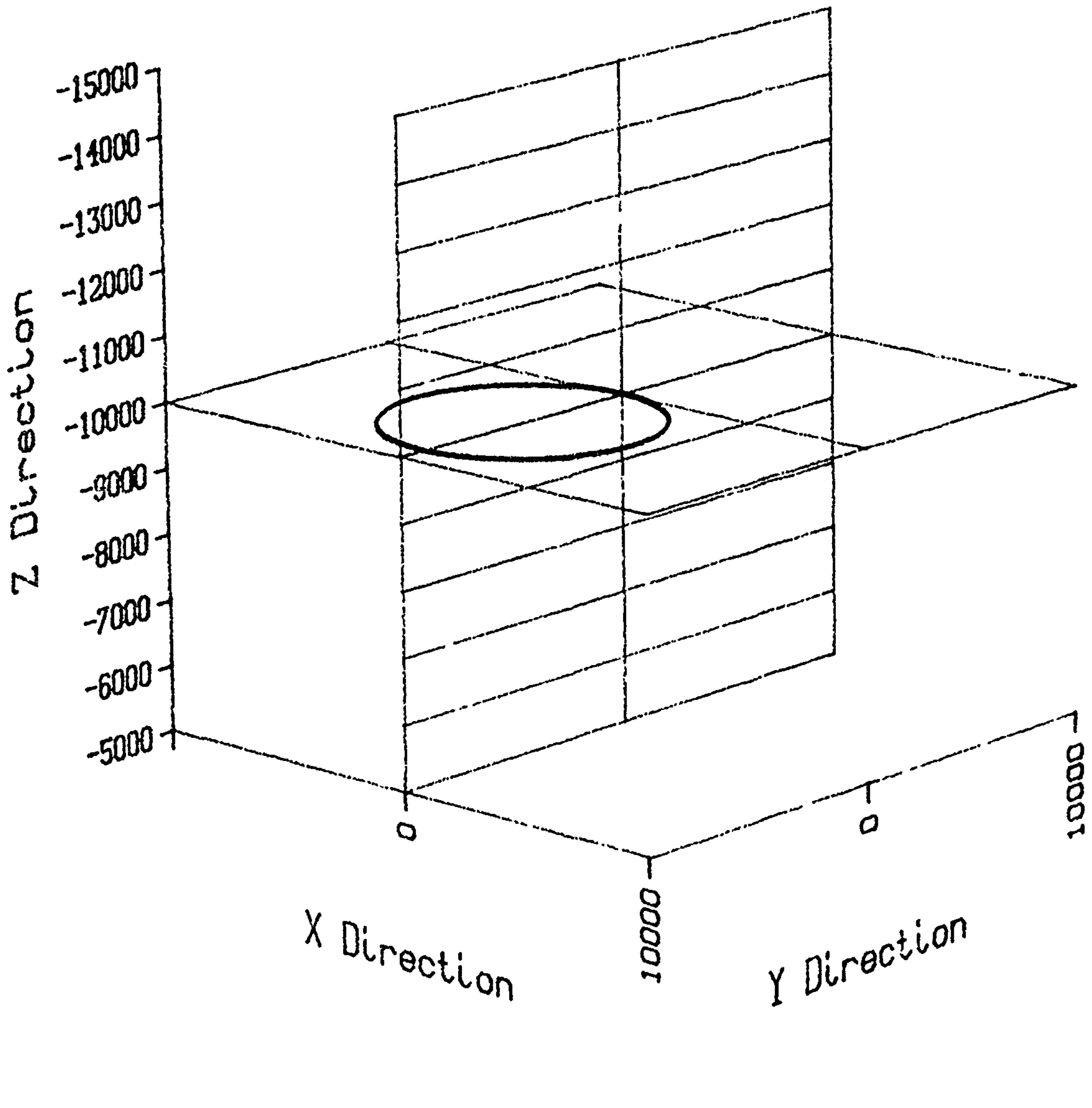
AIRCRAFT FLIGHT PATH



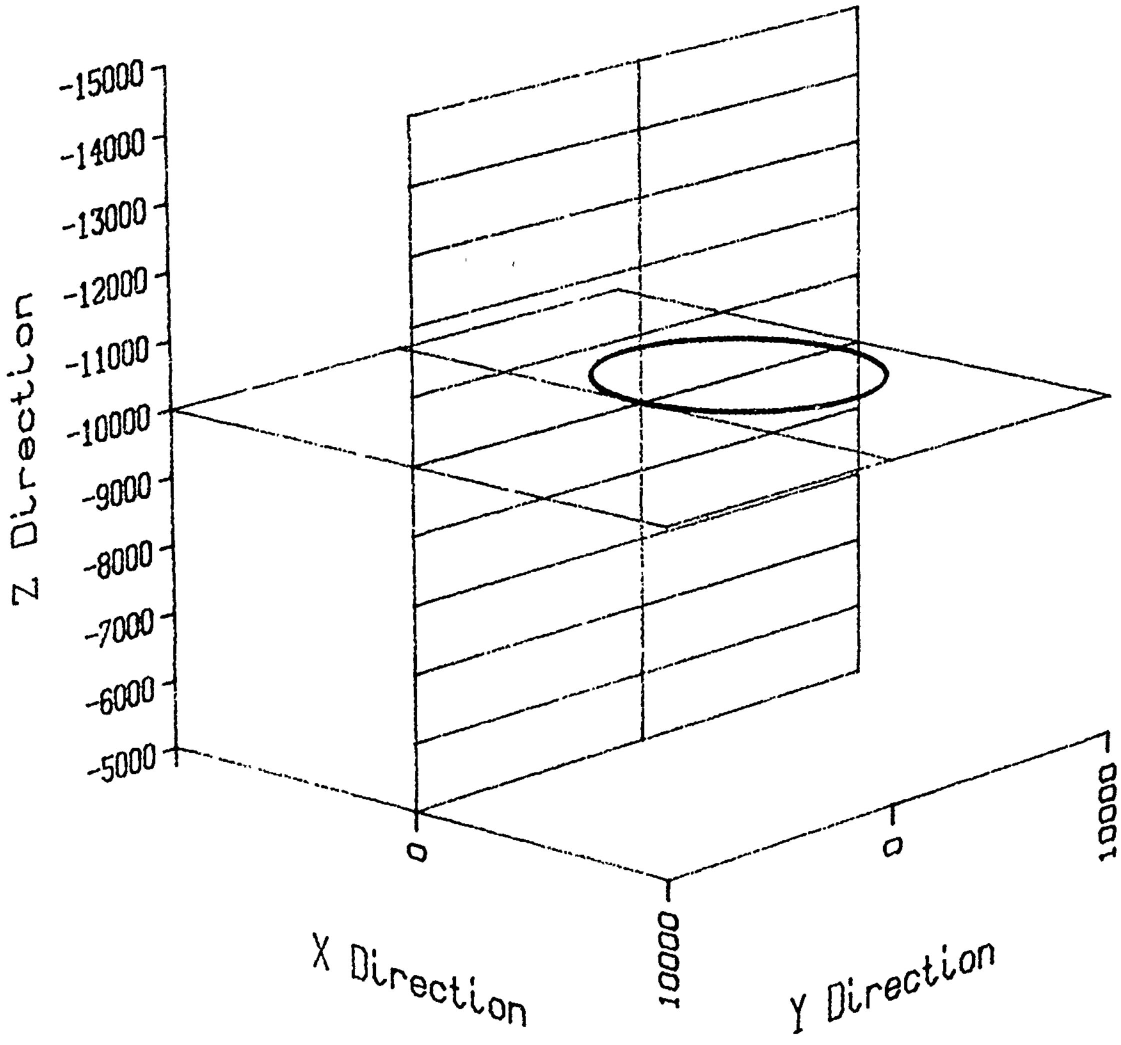
AIRCRAFT FLIGHT PATH



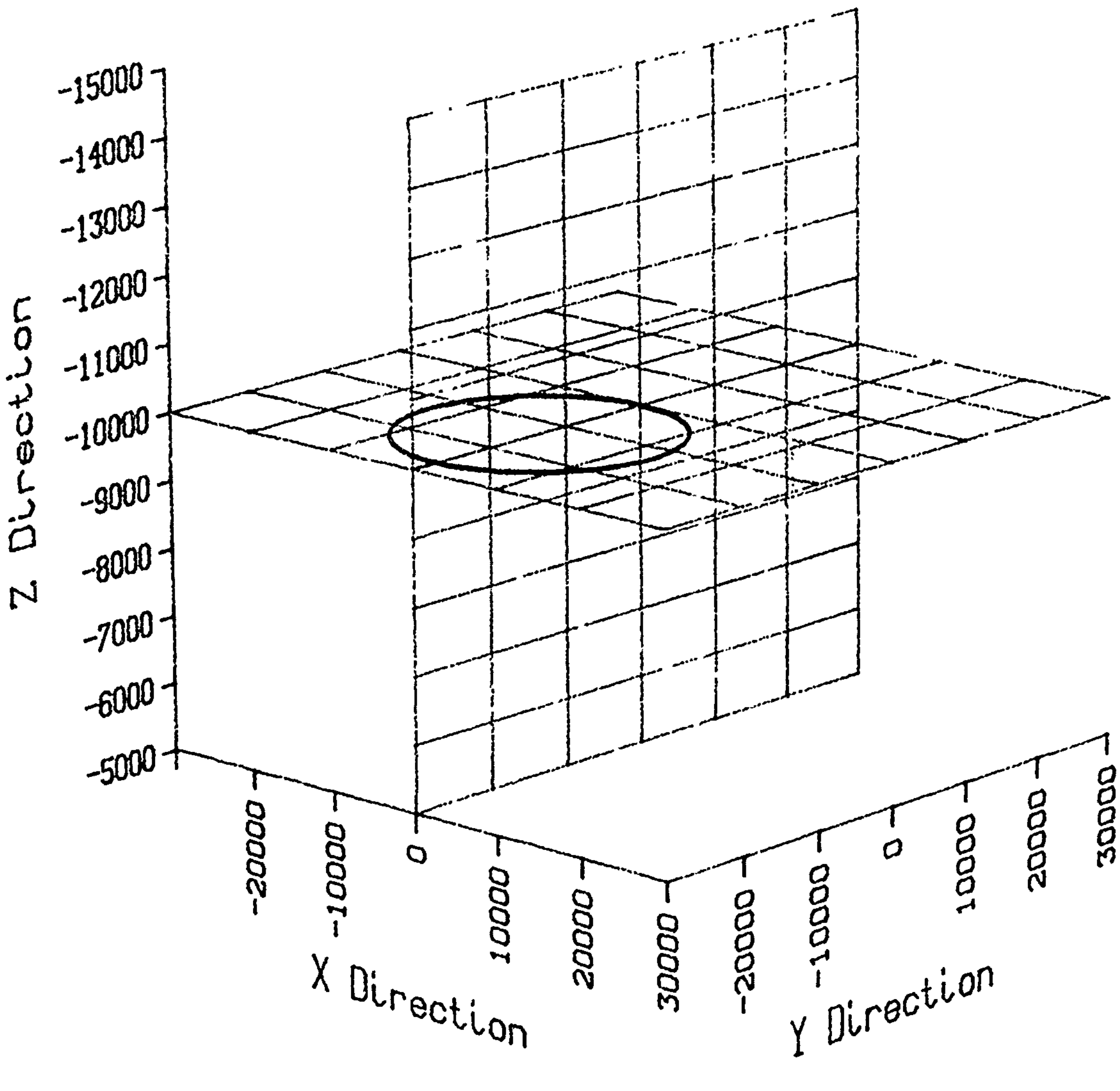
AIRCRAFT FLIGHT PATH



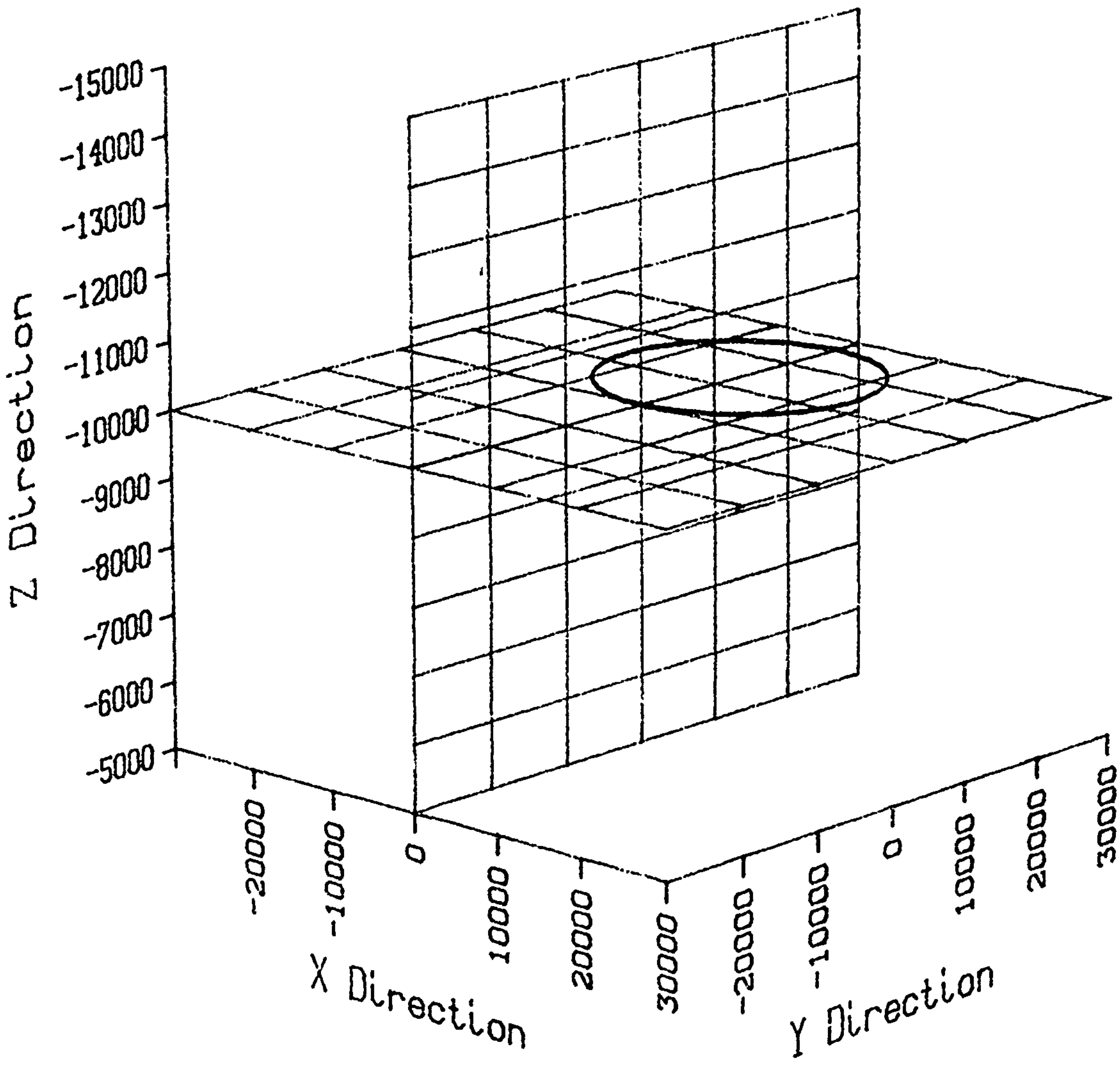
AIRCRAFT FLIGHT PATH



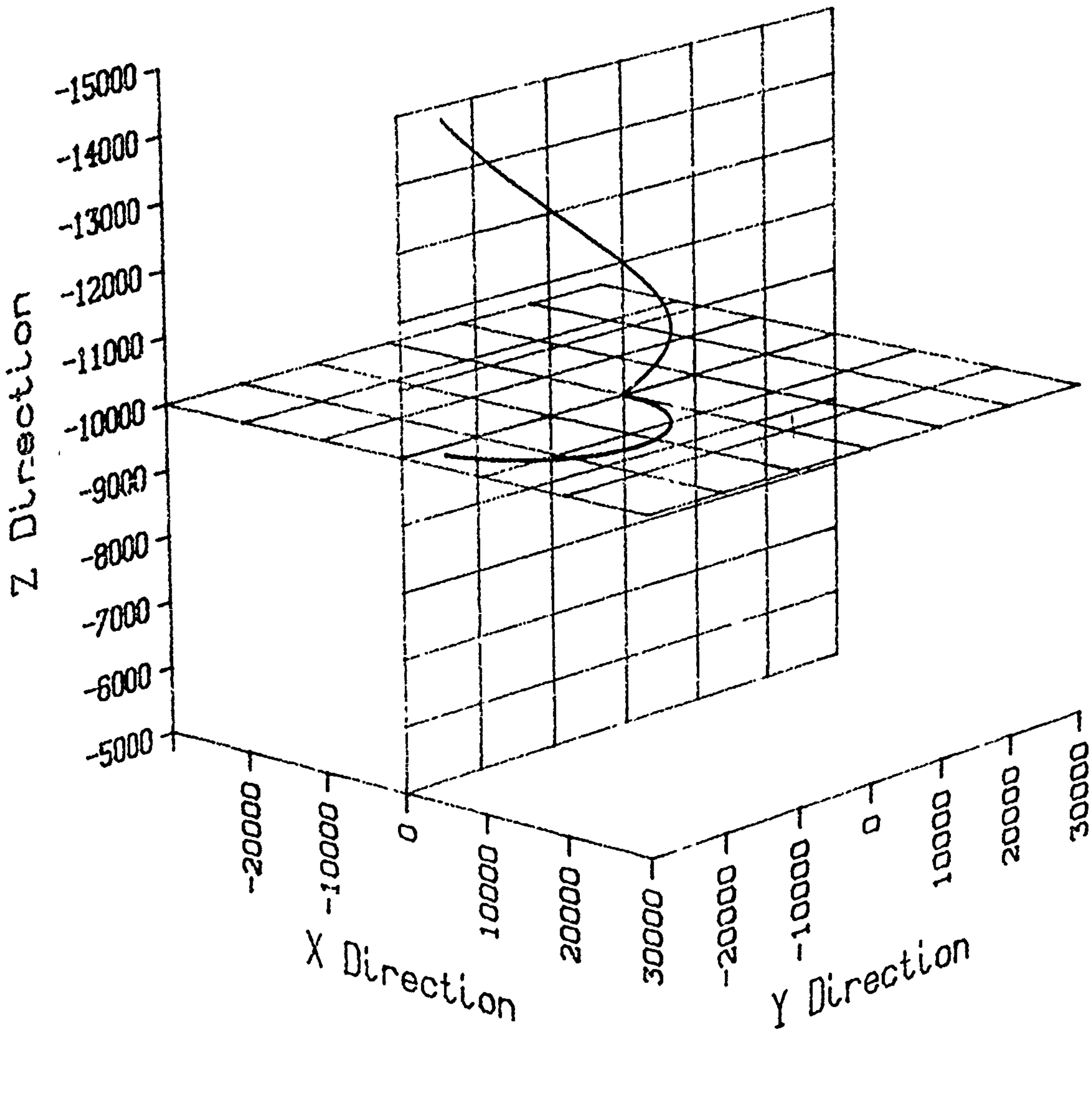
AIRCRAFT FLIGHT PATH



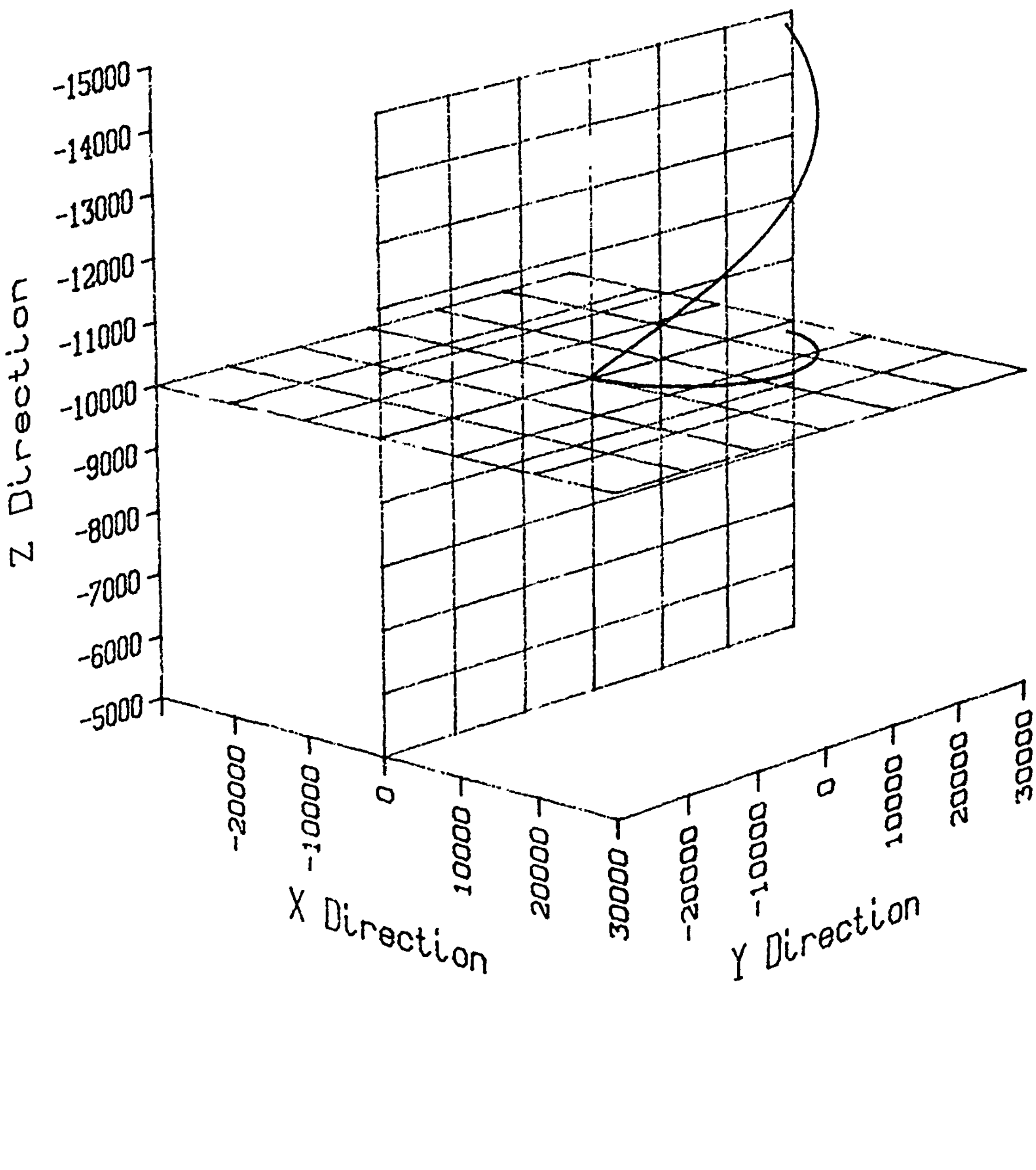
AIRCRAFT FLIGHT PATH



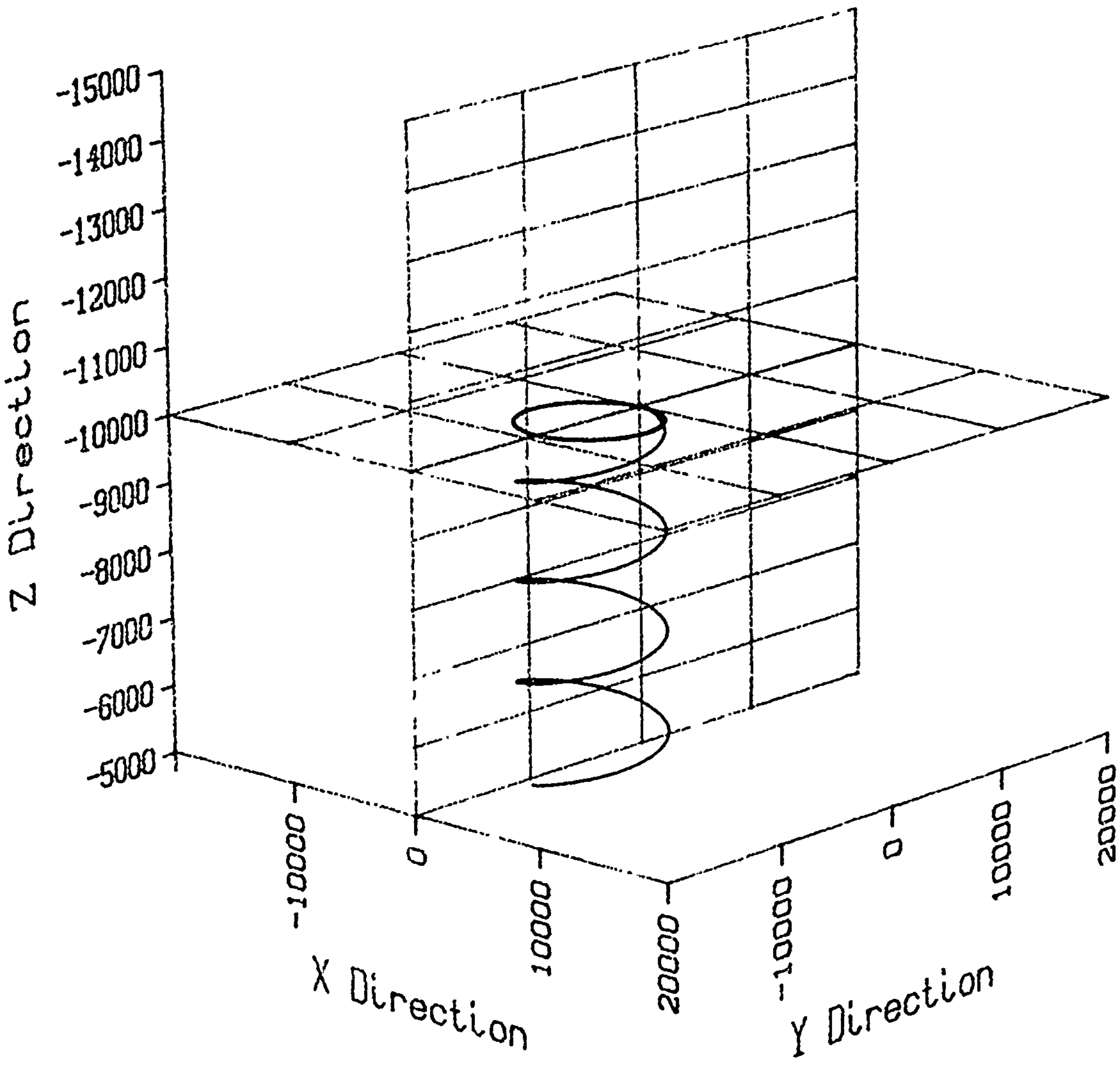
AIRCRAFT FLIGHT PATH



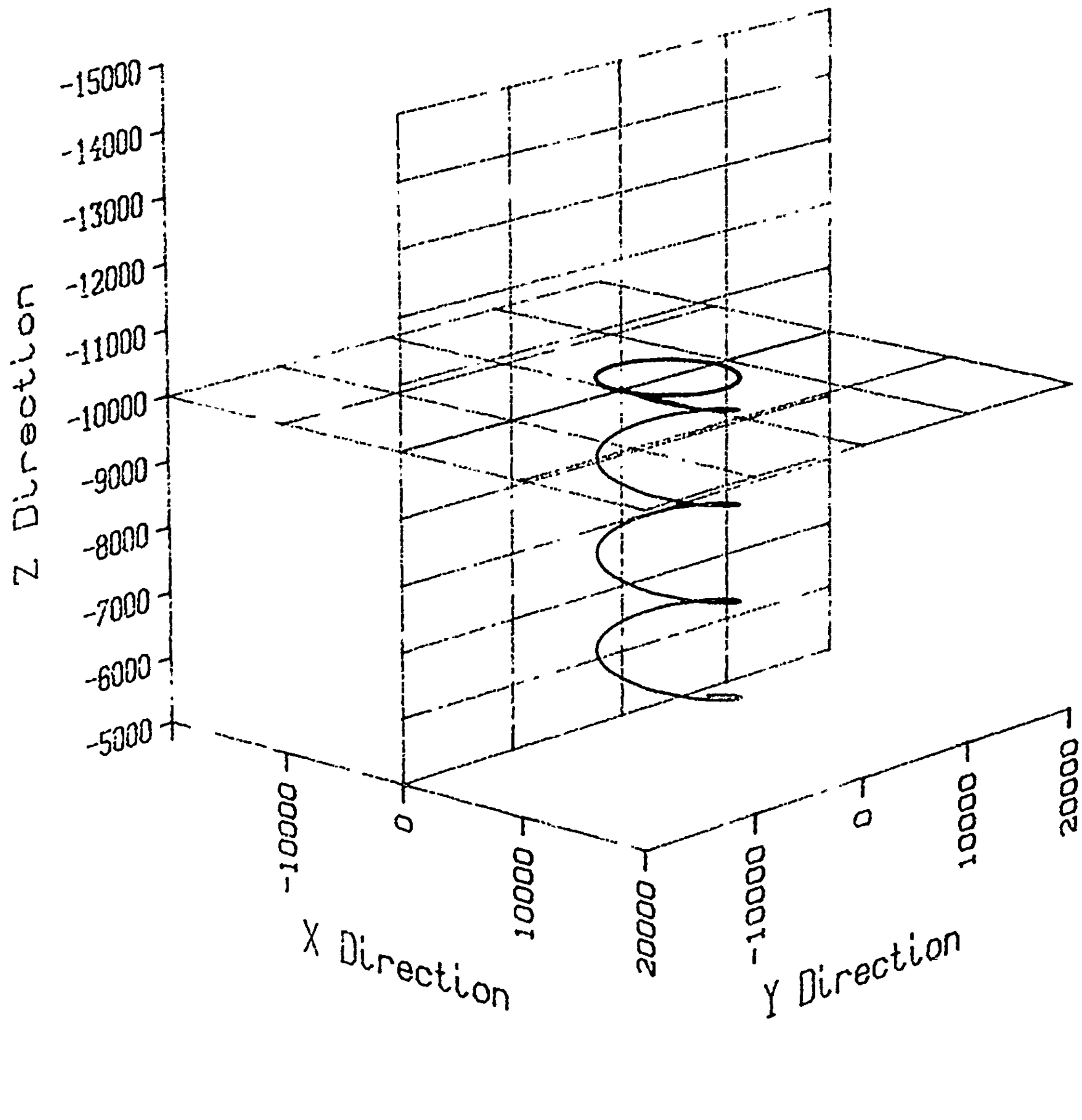
AIRCRAFT FLIGHT PATH



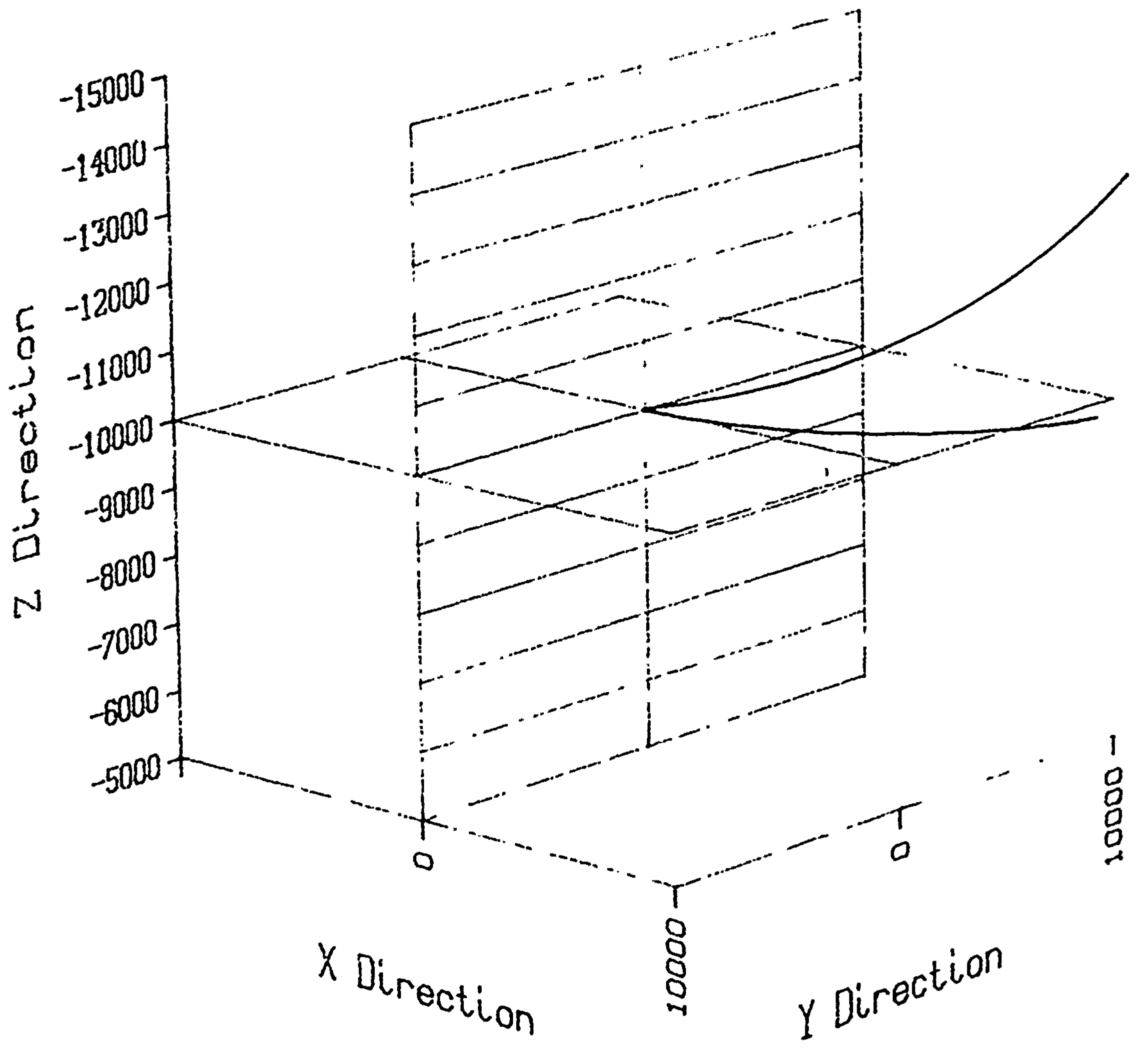
AIRCRAFT FLIGHT PATH



AIRCRAFT FLIGHT PATH



AIRCRAFT FLIGHT PATH



APPENDIX G

```
# file=bg.mak                  make -fbg.mak

MODEL  = m
OBJS   = bg.obj
LIBS   = \tc\lib\graphics \tc\lib\emu \tc\lib\math$(MODEL) \tc\lib\c$(MODEL)
CFLAGS = -c -m$(MODEL) -I\tc\include

bg.exe: $(OBJS)
  tlink \tc\lib\c0$(MODEL) bg, $*, ,$(LIBS)

.c.obj:
  tcc $(CFLAGS) $<
```

```
/* Program bg -- from xbg --   now makes hard copy  */
/* Now ANSI C with prototypes                        */
/* Last modified 4/25/91          cb harmon          */

#include <graphics.h>       /* initgraph, line, closegraph */
                            /* registerbgidriver */
#include <stdlib.h>         /* max */
#include <stdio.h>          /* scanf, printf, sprintf */
#include <dos.h>            /* _stklen, delay */
#include <math.h>           /* log10, floor, pow, fabs */
#include <string.h>         /* strcpy, strcat, outtextxy */

/* extremes of input values */
#define XSMALL +100000000000.0;
#define XLARGE -100000000000.0;
#define YSMALL +100000000000.0;
#define YLARGE -100000000000.0;
/* integer limits of screen */
#define XMIN 0
#define YMIN 0
int XMAX, YMAX;   /* read as input later */
/* assign depth of stack, a global variable */
extern unsigned _stklen = 4000; /* default is 4000 */
struct axis {
        double start, end, tic, major;
        double vstart, vend, vslope;
        int   ivstart, ivend;
} xaxis, yaxis;
/* --------------- Prototypes ---------------------- */
void autoscale(struct axis *px);
void draw_axes(struct axis *px, struct axis *py);
void plot_xy(double xary[], double yary[], int imax,
     struct axis *px, struct axis *py);
void print_axis(struct axis *px);
int dtoi(double d, struct axis *px);
int format(double position);
void dj_graphic(int copies);
/* ------------------- MAIN ------------------------- */
int main(void)
{
        int ncopies, override, driver, mode, npts, i;
        char xtemp[80],xlabel[80],ytemp[80],ylabel[80];
        struct axis *pxaxis, *pyaxis;
        double xin, yin, xa[1001], ya[1001];
        double xmin, xmax, ymin, ymax;
/* get maximum pixel value in x and y */
        scanf("%d %d %d %d",&XMAX,&YMAX,&ncopies,&override);
        printf("XMAX,YMAX,ncopies,override= %6d %6d %4d %4d\n",
                XMAX,YMAX,ncopies,override);
/* read in the x,y array and its attributes */
        scanf("%d",&npts);
        printf("npts= %6d\n",npts);
        xmin = XSMALL;
        xmax = XLARGE;
        ymin = YSMALL;
        ymax = YLARGE;
        for (i=0; i<npts; i++) {
            scanf("%lf %lf",&xin,&yin);
            if (xin > xmax) xmax = xin;
            if (xin < xmin) xmin = xin;
            if (yin > ymax) ymax = yin;
            if (yin < ymin) ymin = yin;
            xa[i]=xin;
            ya[i]=yin;
            printf("x,y= %8.2f %8.2f\n",xa[i],ya[i]);
        }
        printf("xmin,xmax= %8.2f %8.2f\n",xmin,xmax);
        printf("ymin,ymax= %8.2f %8.2f\n",ymin,ymax);
        if (override)
           scanf("%lf %lf %lf %lf",&xmin,&xmax,&ymin,&ymax);
        printf("xmin,xmax= %8.2f %8.2f\n",xmin,xmax);
        printf("ymin,ymax= %8.2f %8.2f\n",ymin,ymax);
/* structure assignments, will be modified by autoscale */
        pxaxis=&xaxis;
        pyaxis=&yaxis;
        pxaxis->start=xmin;
        pxaxis->end  =xmax;
        pyaxis->start=ymin;
        pyaxis->end  =ymax;
        print_axis(pxaxis);
        print_axis(pyaxis);
/* autoscale the axes */
        autoscale(pxaxis);
        autoscale(pyaxis);
        print_axis(pxaxis);
        print_axis(pyaxis);
/* compute the limits of the viewport, derivation by experiment */
        pxaxis->vstart= pxaxis->start - (pxaxis->end - pxaxis->start)*0.3;
        pxaxis->vend  = pxaxis->end   + (pxaxis->end - pxaxis->start)*0.15;
        pyaxis->vstart= pyaxis->start - (pyaxis->end - pyaxis->start)*0.2;
        pyaxis->vend  = pyaxis->end   + (pyaxis->end - pyaxis->start)*0.2;
        pxaxis->ivstart = XMIN;
        pxaxis->ivend   = XMAX;
        pyaxis->ivstart = YMAX;   /* not a mistake */
        pyaxis->ivend   = YMIN;
        pxaxis->vslope  = ((double) (pxaxis->ivend - pxaxis->ivstart))
                                  / (pxaxis->vend -  pxaxis->vstart);
        pyaxis->vslope  = ((double) (pyaxis->ivend - pyaxis->ivstart))
                                  / (pyaxis->vend -  pyaxis->vstart);
/* graph label assignments */
        sprintf(xtemp,"%8.2f %8.2f %8.2f",pxaxis->start,pxaxis->end,
           pxaxis->major);
        sprintf(ytemp,"%8.2f %8.2f %8.2f",pyaxis->start,pyaxis->end,
           pyaxis->major);
        strcpy(xlabel,"X-Axis  min,max,inc= ");
        strcpy(ylabel,"Y-Axis  min,max,inc= ");
        strcat(xlabel,xtemp);
        strcat(ylabel,ytemp);
/* initialize the graph */
        delay(0);   /* calibration */
        registerbgidriver(EGAVGA_driver);
        detectgraph(&driver,&mode);
        printf("driver,mode= %2d %2d\n",driver,mode);
        delay(5000);   /* in milliseconds */
        initgraph(&driver,&mode,"");
        setbkcolor(EGA_BLUE);
        setcolor(EGA_YELLOW);
/* put the x,y labels */
        outtextxy(0,0,ylabel);
        outtextxy(0,3*textheight(ylabel)/2,xlabel);
```

```
/* draw the x,y axes */
        setcolor(EGA_CYAN);
        draw_axes(pxaxis,pyaxis);
/* plot the points   */
        setcolor(EGA_YELLOW);
        plot_xy(xa,ya,npts,pxaxis,pyaxis);
/* dump hardcopy, if requested */
        if (ncopies)
           dj_graphic(ncopies);
        else
           delay(15000);
        closegraph();
        return 0;
}
/* ----------------- Functions ------------------ */
/* autoscale resets axis major,tic,start,end      */
/*   based on next 1,2 or 5 x 10**n                  */
void autoscale(struct axis *px)
{
        double scale, temp, mantissa, exponent;
        double newscale, factor;
        scale = max(fabs(px->start), fabs(px->end));
        temp=log10(scale);
        mantissa=temp-floor(temp);
        exponent=temp-mantissa;
/* scale newscale to 2.0, 5.0, 10.0 */
        if (mantissa<=0.301030) {
           newscale=2.0;
           px->major=newscale/4.0;
           px->tic  =newscale/20.0; }
        else if (mantissa<=0.698970) {
           newscale=5.0;
           px->major=newscale/5.0;
           px->tic  =newscale/25.0; }
        else {
           newscale=10.0;
           px->major=newscale/10.0;
           px->tic  =newscale/20.0; }
/* scale the results to exponent */
        factor=(double) (pow(10.0,exponent));
        px->major=factor*(px->major);
        px->tic  =factor*(px->tic);
        if  (px->end > 0.0)
             px->end = factor * newscale;
        else px->end = 0.0;
        if  (px->start < 0.0)
             px->start = -factor * newscale;
        else px->start = 0.0;
        return;
}

/* ------------------------------------------------ */
void draw_axes(struct axis *px, struct axis *py)
{
        int ix1,ix2,iy1,iy2,ix,iy;
        double x1,x2,y1,y2,x,y,xtic,ytic,x1t,x2t,y1t,y2t;
/* find the end points of both axes */
        x1=px->start;
        x2=px->end;
        y1=py->start;
        y2=py->end;
        ix1 = dtoi(x1,px);
        ix2 = dtoi(x2,px);
        iy1 = dtoi(y1,py);
        iy2 = dtoi(y2,py);
/* draw horiz lines at each major increment */
        y = y1;
        while (y <= y2 + 0.00001*(y2-y1))
        {
           iy = dtoi(y,py);
           line(ix1,iy,ix2,iy);
           y = y + py->major;
        }
/* draw vert lines at each major increment */
        x = x1;
        while (x <= x2 + 0.00001*(x2-x1))
        {
           ix = dtoi(x,px);
           line(ix,iy1,ix,iy2);
           x = x + px->major;
        }
/* draw horiz tic marks */
        xtic = 0.015*(x2-x1);
        x2t = x1 + xtic;
        x1t = x1 - xtic;
        ix1 = dtoi(x1t,px);
        ix2 = dtoi(x2t,px);
        y = y1;
        while (y <= y2 + 0.00001*(y2-y1))
        {
           iy = dtoi(y,py);
           line(ix1,iy,ix2,iy);
           y = y + py->tic;
        }
/* draw vert tic marks */
        ytic = 0.015*(y2-y1);
        y2t = y1 + ytic;
        y1t = y1 - ytic;
        iy1 = dtoi(y1t,py);
        iy2 = dtoi(y2t,py);
        x = x1;
        while (x <= x2 + 0.00001*(x2-x1))
        {
           ix = dtoi(x,px);
           line(ix,iy1,ix,iy2);
           x = x + px->tic;
        }
        return;
}

/* ---------------------------------------------- */
void plot_xy(double xary[], double yary[], int imax,
     struct axis *px, struct axis *py)
{
        int ix1,ix2,iy1,iy2;
        double x1,x2,y1,y2;
        int i;
        for (i=0; i< imax-1; i++) {
            ix1 = dtoi(xary[i],   px);
            ix2 = dtoi(xary[i+1], px);
            iy1 = dtoi(yary[i],   py);
            iy2 = dtoi(yary[i+1], py);
            line(ix1,iy1,ix2,iy2);
        }
        return;
}
/* ---------------------------------------------- */
void print_axis(struct axis *px)
{
        printf("axis %f %f\n",px->start,px->end);
        return;
}
/* ---------------------------------------------- */
/* dtoi computes integer screen pixel coordinate, given its    */
/*   floating point (double) value                             */
int dtoi(double d, struct axis *px)
{
        return (px->ivstart + (int) (px->vslope * (d - px->vstart)));
}
/* ---------------------------------------------- */
int format(double position)
{
        int width = 6;

        if(position < 1000.0)width--;
        if(position < 100.0)width--;
        if(position < 10.0)width--;
        return(width);
}
/* ---------------------------------------------- */
/* hp deskjet EGAVGA screen dump, 75 dpi          */
void dj_graphic(int copies)
{
int i,j,k,p,q,xasp,yasp,
        maxx=getmaxx()+1,
        maxy=getmaxy()+1;
static char graph_ends[]="\x1B*rB";
static char graph_init[]=
                "\x1B""E\x1B&l1H\x1B&l0O\x1B*p0X\x1B*p0Y\x1B*t";
double xprint,yprint,prstep,aspr;
char m,resolution[3];
int ybytes, color, icopy;

for (icopy=0; icopy<copies; icopy++) {
        getaspectratio(&xasp,&yasp);
        aspr=(double)xasp/(double)yasp;
        setviewport(0,0,maxx,maxy,0);

        xprint=1000.0;   /* x always starts here */
        yprint=1000.0;   /* first initial y      */
        strcpy(resolution,"75");                 /* 75 dpi */
        prstep=9.6*aspr;            /* adjust to match screen size */
        fprintf(stdprn,"%s%sR",graph_init,resolution);

        ybytes=(int)((maxy+4)/8);
        /* position the print head */
        fprintf(stdprn,"\x1B&a%-*.1fh%-*.1fV",
                                format(xprint),xprint,
                                format(yprint),yprint);
        fprintf(stdprn,"\x1B*r1A");  /* initialize printer */
        /* outer loop -- rows */
        for(j=0;j<maxx;j++)
        {
                /* position the print head */
                fprintf(stdprn,"\x1B&a%-*.1fh%-*.1fV",
                                        format(xprint),xprint,
                                        format(yprint),yprint);
                yprint+=prstep;
                fprintf(stdprn,"\x1B*b%dW",ybytes); /* prepare to send */
                /* inner loop -- columns */
                for(i=0;i<ybytes;i++)
                {
                        m=0;
                        for(k=0;k<8;k++)
                        {
                                m<<=1;
                                color=getpixel(maxx-j,i*8+k);
                                if(color!=0)m++; /* 0 is background */
                        }

                /* repair for cntlZ byte not handled correctly thru stdprn */
                        if(m=='\x1A')
                        {
                                m='\x18';
                        }
                        fprintf(stdprn,"%c",m); /* send 8 pixels */
                }
        }
        fprintf(stdprn,"%s",graph_ends);

        fprintf(stdprn,"\x1B&l%dX",1);
        fprintf(stdprn,"\x0C\x1B""E"); /* formfeed and reset */
}
}
```

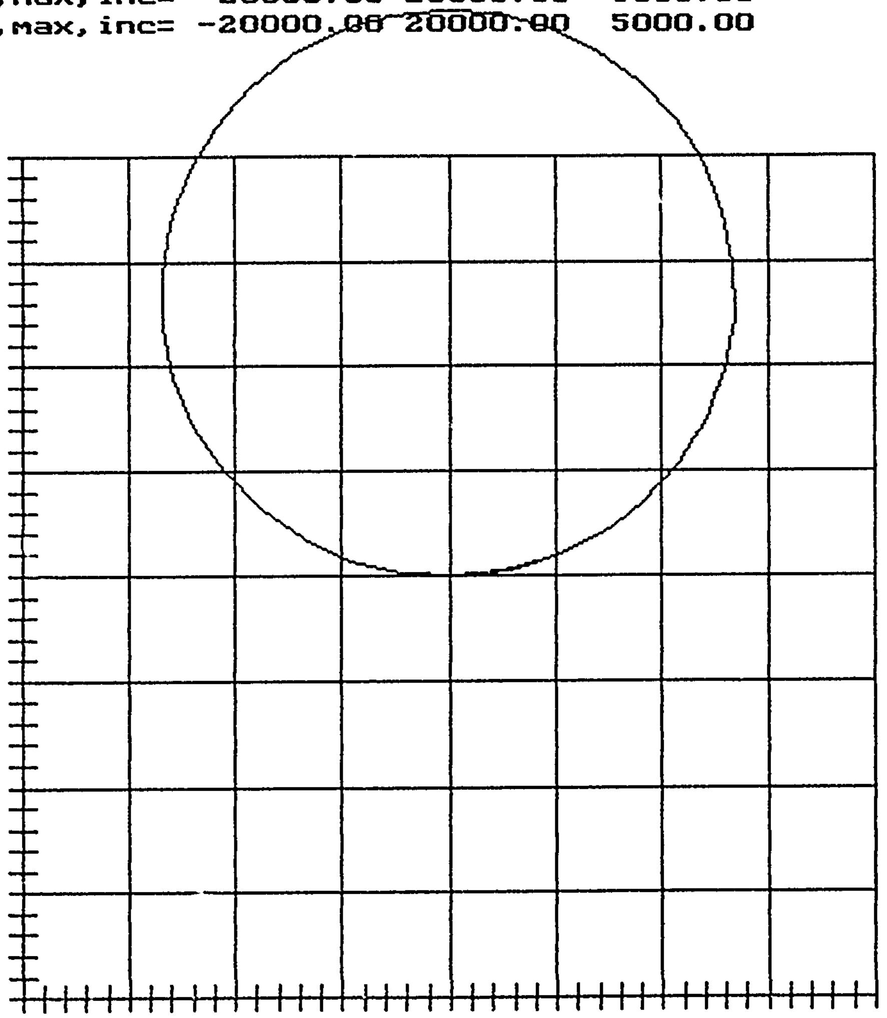
APPENDIX H

```
To filter the output from fs to get the form for bg,
   (1) delete the top records to produce INFILE, a file of
         records with the same number of fields in every record.
   (2) run utility awk as follows
         awk '{ printf "%8d %8d\n", $2, $3}' INFILE >OUTFILE
           to get a file of records with two fields in every record.
           This example produces only fields 2 and 3.  You may be
             interested in a different two fields.  The new file is
               OUTFILE.  Remember the number of records in it now.
   (3) edit OUTFILE to put these two records in front of the rest
            479 479 0 1         or         479 479 1 1 (for hardcopy)
            X
         where X is the number of records remaining after those two.
         Then add a record to the end of the file.  This record should
           contain four numbers:
              (1) min value (approx) in column 1,
              (2) max value (approx) in column 1,
              (3) min value (approx) in column 2, and
              (4) max value (approx) in column 2.
           The resulting graph will contain col 1 numbers on the abscissa
           and col 2 numbers on the ordinate.  The program "autoscales"
           and "corrects" the min/max values specified using the 2-5-10
           rule.  For example, if col 1 contains a min value of -18000
           and a max of 9000, the bg program will scale the abscissa to
           accommodate -20000 through +20000.  Autoscale is applied to
           each axis independently, allowing psi vs z, for example.

To execute, bg <OUTFILE
   Files INFILE.EG and OUTFILE.EG are provided as examples.
```

```
  0.00      0      0  -10000  500  500    0   0    0   0  30  41569  2934
  5.00   2486    232  -10000  500  491   92   0   11  -0  30  41569  2934
 10.00   4886    919  -10000  500  466  182   0   21  -0  30  41569  2934
 15.00   7118   2038  -10000  500  424  264   0   32  -0  30  41569  2934
 20.00   9105   3549  -10000  500  368  338   0   43  -0  30  41569  2934
 25.00  10779   5402  -10000  500  299  400   0   53  -0  30  41569  2934
 30.00  12082   7532  -10000  500  220  449   0   64  -0  30  41569  2934
 35.00  12969   9866  -10000  500  134  482  -0   74   0  30  41569  2934
 40.00  13410  12323  -10000  500   42  498  -0   85   0  30  41569  2934
 45.00  13390  14820  -10000  500  -50  498  -0   96   0  30  41569  2934
 50.00  12909  17271  -10000  500 -141  480  -0  106   0  30  41569  2934
 55.00  11984  19591  -10000  500 -227  445  -0  117   0  30  41569  2934
 60.00  10647  21700  -10000  500 -306  396  -0  128   0  30  41569  2934
 65.00   8943  23526  -10000  500 -374  333   0  138  -0  30  41569  2934
 70.00   6931  25006  -10000  500 -429  258   0  149  -0  30  41569  2934
 75.00   4680  26089  -10000  500 -469  174   0  160  -0  30  41569  2934
 80.00   2269  26738  -10000  500 -493   85   0  170  -0  30  41569  2934
 85.00   -222  26931  -10000  500 -500   -8   0  181  -0  30  41569  2934
 90.00  -2705  26661  -10000  500 -490 -100   0  192  -0  30  41569  2934
 95.00  -5096  25937  -10000  500 -463 -189   0  202  -0  30  41569  2934
100.00  -7312  24784  -10000  500 -421 -271  -0  213   0  30  41569  2934
105.00  -9278  23242  -10000  500 -363 -344  -0  223   0  30  41569  2934
110.00 -10925  21364  -10000  500 -294 -405  -0  234   0  30  41569  2934
115.00 -12197  19214  -10000  500 -214 -452  -0  245   0  30  41569  2934
120.00 -13051  16866  -10000  500 -127 -484  -0  255   0  30  41569  2934
125.00 -13456  14400  -10000  500  -35 -499  -0  266   0  30  41569  2934
130.00 -13400  11902  -10000  500   57 -497   0  277  -0  30  41569  2934
135.00 -12884   9458  -10000  500  148 -478   0  287  -0  30  41569  2934
140.00 -11925   7150  -10000  500  234 -442   0  298  -0  30  41569  2934
145.00 -10557   5059  -10000  500  312 -392   0  308  -0  30  41569  2934
150.00  -8827   3256  -10000  500  378 -328   0  319  -0  30  41569  2934
155.00  -6794   1803  -10000  500  432 -252   0  330  -0  30  41569  2934
160.00  -4527    750  -10000  500  472 -168   0  340  -0  30  41569  2934
165.00  -2105    133  -10000  500  494  -78  -0  351   0  30  41569  2934
170.00    389    -26  -10000  500  500   14  -0    2   0  30  41569  2934
175.00   2870    278  -10000  500  489  106  -0   12   0  30  41569  2934
180.00   5252   1034  -10000  500  461  195  -0   23   0  30  41569  2934
```

```
479 479 1 1
37
         0         0
      2486       232
      4886       919
      7118      2038
      9105      3549
     10779      5402
     12082      7532
     12969      9866
     13410     12323
     13390     14820
     12909     17271
     11984     19591
     10647     21700
      8943     23526
      6931     25006
      4680     26089
      2269     26738
      -222     26931
     -2705     26661
     -5096     25937
     -7312     24784
     -9278     23242
    -10925     21364
    -12197     19214
    -13051     16866
    -13456     14400
    -13400     11902
    -12884      9458
    -11925      7150
    -10557      5059
     -8827      3256
     -6794      1803
     -4527       750
     -2105       133
       389       -26
      2870       278
      5252      1034
-19000.0 19000.0 -19000.0 19000.0
```

```
Y-Axis   min,max,inc= -20000.00 20000.00  5000.00
X-Axis   min,max,inc= -20000.00 20000.00  5000.00
```



APPENDIX I

This directory is to develop the flight simulator
program in Ada.

```
------------------
acs create library [.fslib]        # just once
acs set library [.fslib]           # each login to vax
ada mat_math.ads                   # if it has changed
ada mat_math.adb                   # ditto
ada mat_t.ada                      # test driver for package mat_math
acs link mat_t
run /output=<outfile> mat_t
ada fs.ada                         # main subprogram
acs link fs
run /input=<infile> /output=<outfile> fs
------------------
acs delete library [.fslib]
del fs.exe;*
del mat_t.exe;*
```

```
cat arci.bat ast ..\vaxadafs\fs.ada ast ..\vaxadafs\mat_math.ads ast ..\vaxadafs

*********************************************************************

-- source = fs.ada, flight simulation, last=8/28/91
-- main procedure for 3 degree of freedom flt sim
-- See also mat_math.ads mat_math.adb

with text_io; with mat_math; use mat_math;
with math_lib;

procedure fs is
        package int_io is new text_io.integer_io(integer);
        package real_io is new text_io.float_io(real);
        package real_math is new math_lib(real);
use real_math;
        DTR :     constant real:=     57.29577951;
        EE  :     constant real:=      2.71828183;
        PI  :     constant real:=      3.14159265;
        TWOPI :   constant real:=      6.28318531;
        iprint, idebug, iconas, nptscl, imain, i, j                 : integer;
        tmin, tmax, dt, cd0, aspect, eff, swing, weight, thrust : real;
        alt, aspd, psi, thet, phi, tclmin, tclmax, dtcl, frac   : real;
        p, q, r, omegas, psidot, thetdot, phidot, omegae        : real;
        psib, thetb, phib, t, temp, qbar, rho, mass, lift, drag : real;
        tclary:array(1..1000) of real;
        clary: array(1..1000) of real;
        wt1,  wt2,  wt3,  wts                 :vector;
        udot1,udot2,udot3,udots               :vector;
        xdot1,xdot2,xdot3,xdots, x1           :vector;
        apsi,athet,aphi,aipsi,aithet,aiphi    :matrix;
begin
-- get simulation parameters
        real_io.get(tmin);
        real_io.get(tmax);
        real_io.get(dt);
        int_io.get(iprint);
        int_io.get(idebug);
        int_io.get(iconas);
--
        text_io.put("tmin,tmax,dt,iprint,idebug,iconas=");
        real_io.put(tmin,7,4,0);
        real_io.put(tmax,7,4,0);
        real_io.put(dt,  7,4,0);
        int_io.put(iprint,4);
        int_io.put(idebug,4);
        int_io.put(iconas,4);
        text_io.new_line;
-- get aerodynamics parameters
        real_io.get(cd0);
        real_io.get(aspect);
        real_io.get(eff);
        real_io.get(swing);
        real_io.get(weight);
        mass:=  weight / 32.174;
        real_io.get(thrust);
--
        text_io.put("cd0,aspect,eff=");
        real_io.put(cd0,    3,4,0);
        real_io.put(aspect, 3,4,0);
```

```
       real_io.put(eff,    3,4,0);
       text_io.new_line;
       text_io.put("swing,weight,thrust=");
       real_io.put(swing,  12,0,0);
       real_io.put(weight, 12,0,0);
       real_io.put(thrust, 12,4,0);
       text_io.new_line;
-- get initial conditions
       real_io.get(alt);
       real_io.get(aspd);
       rho:=   0.0023769 * exp(-alt / 30500.0);
       qbar:=  0.5 * rho * aspd * aspd;
       real_io.get(psi);
       real_io.get(thet);
       real_io.get(phi);
--
       text_io.put("alt,aspd,psi,thet,phi=");
       real_io.put(alt,    12,0,0);
       real_io.put(aspd,    8,0,0);
       real_io.put(psi,     3,4,0);
       real_io.put(thet,    3,4,0);
       real_io.put(phi,     3,4,0);
       text_io.new_line;
       text_io.put("mass,rho,qbar=");
       real_io.put(mass,    7,2,0);
       real_io.put(rho,     5,8,0);
       real_io.put(qbar,    7,4,0);
       text_io.new_line;
-- get cl vs time curve
       real_io.get(tclmin);
       real_io.get(dtcl);
       int_io.get(nptscl);
       text_io.put("tclmin,dtcl,nptscl=");
       real_io.put(tclmin,  5,2,0);
       real_io.put(dtcl,    5,2,0);
       int_io.put(nptscl, 6);
       text_io.new_line;
       tclmax := tclmin + dtcl * real(nptscl - 1);
       for j in 1..nptscl loop
          tclary(j) := tclmin + dtcl * real(j - 1);
          real_io.get(temp);
          clary(j) := temp;
          real_io.put(tclary(j), 5,2,0);
          real_io.put( clary(j), 3,6,0);
          text_io.new_line;
       end loop;
-- initializations
       t:=   tmin;
--
       if t < tclmin or t > tclmax then
          text_io.put("Error in getcl,  t,tclmin,tclmax = ");
          real_io.put(t,     6,3,0);
          real_io.put(tclmin,6,3,0);
          real_io.put(tclmax,6,3,0);
          text_io.new_line;
          temp := 1.0;
       else
          for i in 1..nptscl loop
              j := i;
              exit when t < tclary(j);
```

```
            end loop;
            frac := (t - tclary(j-1)) / dtcl;
            temp := clary(j-1) + frac * (clary(j) - clary(j-1));
         end if;
--
         frac:= 0.5;
         lift:= qbar * swing * temp;
         drag:= qbar * swing * (cd0 + temp*temp/(PI*aspect*eff));
         p:=0.0;
         q:=0.0;
         r:=0.0;
         psidot:= 0.0;
         thetdot:=0.0;
         phidot:= 0.0;
         psib  := psi;
         thetb := thet;
         phib  := phi;
         wt1(1)  := 0.0;
         wt1(2)  := 0.0;
         wt1(3)  := weight;
         x1(1)   := 0.0;
         x1(2)   := 0.0;
         x1(3)   := -alt;
         xdots(1)  := aspd;
         xdots(2)  := 0.0;
         xdots(3)  := 0.0;
-- get the transform matrices based on Euler angles
         get_mats(psi,thet,phi,apsi,athet,aphi);
         minv(apsi,aipsi);
         minv(athet,aithet);
         minv(aphi,aiphi);
-- convert stability xdots vector to inertial xdot1
         mvmult(aphi, xdots,xdot3);
         mvmult(athet,xdot3,xdot2);
         mvmult(apsi, xdot2,xdot1);
-- convert weight vector wt1 to stability wts
         mvmult(aipsi,wt1,wt2);
         mvmult(aithet,wt2,wt3);
         mvmult(aiphi,wt3,wts);
-- header records
         text_io.put("fs program begins");
         text_io.new_line;
         text_io.put("   time        x        y        z   aspd   xdot");
         text_io.put("   ydot   zdot    psi   thet    phi     lift    drag");
         text_io.new_line(2);
--
         imain := 0;
-- main loop
--
while t <= tmax loop
-- main output record
         if imain = 0 then
            real_io.put(t,4,2,0);
            real_io.put(x1(1),7,0,0);
            real_io.put(x1(2),7,0,0);
            real_io.put(x1(3),8,0,0);
            real_io.put(aspd,5,0,0);
            real_io.put(xdot1(1),5,0,0);
            real_io.put(xdot1(2),5,0,0);
            real_io.put(xdot1(3),5,0,0);
```

```
          real_io.put(psi*DTR,5,0,0);
          real_io.put(thet*DTR,5,0,0);
          real_io.put(phi*DTR,5,0,0);
          real_io.put(lift,7,0,0);
          real_io.put(drag,6,0,0);
          text_io.new_line;
       end if;
-- stop if below sea level
       exit when alt < 0.0;
-- find the forces acting on the body
       if t < tclmin or t > tclmax then
          text_io.put("Error in getcl,  t,tclmin,tclmax = ");
          real_io.put(t,     6,3,0);
          real_io.put(tclmin,6,3,0);
          real_io.put(tclmax,6,3,0);
          text_io.new_line;
          temp := 1.0;
       else
          for i in 1..nptscl loop
             j := i;
             exit when t < tclary(j);
          end loop;
          frac := (t - tclary(j-1)) / dtcl;
          temp := clary(j-1) + frac * (clary(j) - clary(j-1));
       end if;
--
       lift:= qbar * swing * temp;
       drag:= qbar * swing * (cd0 + temp*temp/(PI*aspect*eff));
       if idebug > 0 then
          text_io.put("j,nptscl,frac,temp,lift,drag=");
          int_io.put(j,4);
          int_io.put(nptscl,4);
          real_io.put(frac,3,6,0);
          real_io.put(temp,3,6,0);
          real_io.put(lift,8,0,0);
          real_io.put(drag,8,0,0);
          text_io.new_line;
       end if;
-- get the transform matrices based on Euler angles
       get_mats(psi,thet,phi,apsi,athet,aphi);
       minv(apsi,aipsi);
       minv(athet,aithet);
       minv(aphi,aiphi);
-- convert stability xdots vector to inertial xdot1
       mvmult(aphi, xdots,xdot3);
       mvmult(athet,xdot3,xdot2);
       mvmult(apsi, xdot2,xdot1);
-- convert weight vector wt1 to stability wts
       mvmult(aipsi,wt1,wt2);
       mvmult(aithet,wt2,wt3);
       mvmult(aiphi,wt3,wts);
-- using eq 2.19 compute linear accel (f=ma), stab axes
       udots(1):=  (wts(1) - drag + thrust) / mass;
       udots(2):=   wts(2) / mass;
       udots(3):=  (wts(3) - lift) / mass;
-- if constant airspeed profile, then set tangential accel to zero
       if iconas > 0  then udots(1) := 0.0; end if;
-- aspd increment entirely due to udots(1)
       aspd := aspd + dt * udots(1);
-- convert stability udots to inertial udot1
```

```
        mvmult(aphi,udots,udot3);
        mvmult(athet,udot3,udot2);
        mvmult(apsi,udot2,udot1);
        if idebug > 0 then
           text_io.put("udots=  ");
           real_io.put(udots(1),6,2,0);
           real_io.put(udots(2),6,2,0);
           real_io.put(udots(3),6,2,0);
           text_io.new_line;
           text_io.put("udot1=  ");
           real_io.put(udot1(1),6,2,0);
           real_io.put(udot1(2),6,2,0);
           real_io.put(udot1(3),6,2,0);
           text_io.new_line;
        end if;
-- advance xdot1 then x1
        for j in 1..3 loop
               xdot1(j):=xdot1(j) + dt*udot1(j);
               x1(j)    :=   x1(j) + dt*xdot1(j);
        end loop;
--  geometry to get psi, thet, (phi constrained to initial value)
        psi  := atan( xdot1(2) / xdot1(1) );
        if xdot1(1) < 0.0 then
           psi := psi + PI;
        else
           if xdot1(2) < 0.0 then psi := psi + TWOPI; end if;
        end if;
        psidot := (psi - psib) / dt;
        if (psi-psib) > PI then psidot := 0.0; end if;
        if (psib-psi) > PI then psidot := 0.0; end if;
--
        temp := sqrt(xdot1(1)*xdot1(1) + xdot1(2)*xdot1(2));
        thet := atan(-xdot1(3) / temp);
        thetdot := (thet - thetb) / dt;
        phidot  := 0.0;
-- back values for psi,thet,phi
        psib  := psi;
        thetb := thet;
        phib  := phi;
-- eq 2-46 to get p,q,r from psidot, thetdot, phidot
        p := phidot - psidot * sin(thet);
        q := thetdot * cos(phi) + psidot * cos(thet) * sin(phi);
        r :=-thetdot * sin(phi) + psidot * cos(thet) * cos(phi);
        omegas := sqrt(p*p + q*q + r*r);
        omegae := sqrt(psidot*psidot + thetdot*thetdot + phidot*phidot);
        if idebug > 0 then
           text_io.put("Eudots= ");
           real_io.put( psidot*DTR,4,4,0);
           real_io.put(thetdot*DTR,4,4,0);
           real_io.put( phidot*DTR,4,4,0);
           real_io.put( omegas*DTR,4,4,0);
           text_io.new_line;
           text_io.put("p,q,r = ");
           real_io.put(p*DTR,4,4,0);
           real_io.put(q*DTR,4,4,0);
           real_io.put(r*DTR,4,4,0);
           real_io.put( omegae*DTR,4,4,0);
           text_io.new_line;
        end if;
-- update the rest
```

```
        alt:=   -x1(3);
        rho:=   0.0023769 * exp(-alt / 30500.0);
        qbar:=  0.5 * rho * aspd * aspd;
--
        t:=     t + dt;
        imain:= imain + 1;
        if imain >= iprint then imain:=0; end if;
end loop;
end fs;

*********************************************************************

-- source = mat_math.ads 8/91 cb harmon for flight sim.
-- specification for a package of matrix math procedures
-- these are tested using the module mat_t
-- vectors have 3 elements; matrices have 3x3

with text_io; with math_lib;
package mat_math is
        type real is digits 9;
        package real_io is new text_io.float_io(real);
        package int_io is new text_io.integer_io(integer);
        package real_math is new math_lib(real);
        type vector is array(1..3) of real;
        type matrix is array(1..3,1..3) of real;

procedure mvmult(a:in matrix; b:in vector; c:out vector);
-- multiplies matrix a by vector b to get vector c

procedure mmmult(a,b:in matrix; c:out matrix);
-- multiplies matrix a by matrix b to get matrix c

function det(a:in matrix) return real;
-- finds the determinant of matrix a

procedure minv(a:in matrix; b:out matrix);
-- finds the inverse of matrix a

procedure get_mats(psi,thet,phi:in real; apsi,athet,aphi:out matrix);
-- gets the rotation matrices for inertial to stability axis

end mat_math;

*********************************************************************

-- source = mat_math.adb 8/91 cb harmon for flight sim.
-- body for a package of matrix math procedures
-- these are tested using the module mat_t
-- vectors have 3 elements; matrices have 3x3

package body mat_math is

procedure mvmult(a:in matrix; b:in vector; c:out vector) is
-- multiplies matrix a by vector b to get vector c
        i:integer;
begin
        for i in 1..3 loop
                c(i):=a(i,1)*b(1) + a(i,2)*b(2) + a(i,3)*b(3);
        end loop;
end mvmult;
```

```
procedure mmmult(a,b:in matrix; c:out matrix) is
-- multiplies matrix a by matrix b to get matrix c
        i,j:integer;
begin
        for i in 1..3 loop
        for j in 1..3 loop
                c(i,j):=a(i,1)*b(1,j)+a(i,2)*b(2,j)+a(i,3)*b(3,j);
        end loop;
        end loop;
end mmmult;

function det(a:in matrix) return real is
-- finds the determinant of matrix a
        d:real;
begin
        d:=  a(1,1)*(a(2,2)*a(3,3)-a(3,2)*a(2,3))
            -a(1,2)*(a(2,1)*a(3,3)-a(3,1)*a(2,3))
            +a(1,3)*(a(2,1)*a(3,2)-a(3,1)*a(2,2));
        return d;
end det;

procedure minv(a:in matrix; b:out matrix) is
-- finds the inverse of matrix a
        d:real;
begin
        d:=det(a);
        b(1,1):= (a(2,2)*a(3,3)-a(3,2)*a(2,3))/d;
        b(1,2):=-(a(1,2)*a(3,3)-a(3,2)*a(1,3))/d;
        b(1,3):= (a(1,2)*a(2,3)-a(2,2)*a(1,3))/d;
        b(2,1):=-(a(2,1)*a(3,3)-a(3,1)*a(2,3))/d;
        b(2,2):= (a(1,1)*a(3,3)-a(3,1)*a(1,3))/d;
        b(2,3):=-(a(1,1)*a(2,3)-a(2,1)*a(1,3))/d;
        b(3,1):= (a(2,1)*a(3,2)-a(3,1)*a(2,2))/d;
        b(3,2):=-(a(1,1)*a(3,2)-a(3,1)*a(1,2))/d;
        b(3,3):= (a(1,1)*a(2,2)-a(2,1)*a(1,2))/d;
exception
        when numeric_error | constraint_error=>
                text_io.put("numeric_error in procedure minv");
                text_io.new_line;
                text_io.put("det(a)= ");
                real_io.put(d);
                text_io.new_line(2);
end minv;

procedure get_mats(psi,thet,phi:in real; apsi,athet,aphi:out matrix) is
-- gets the rotation matrices for inertial to stability axis
-- inputs psi,thet,phi are in radians
-- see derivation in roskam chap 2, p27.
begin
        apsi(1,1):= real_math.cos(psi);
        apsi(1,2):=-real_math.sin(psi);
        apsi(1,3):=0.0;
        apsi(2,1):= real_math.sin(psi);
        apsi(2,2):= real_math.cos(psi);
        apsi(2,3):=0.0;
        apsi(3,1):=0.0;
        apsi(3,2):=0.0;
        apsi(3,3):=1.0;
--
```

```
        athet(1,1):= real_math.cos(thet);
        athet(1,2):=0.0;
        athet(1,3):= real_math.sin(thet);
        athet(2,1):=0.0;
        athet(2,2):=1.0;
        athet(2,3):=0.0;
        athet(3,1):=-real_math.sin(thet);
        athet(3,2):=0.0;
        athet(3,3):= real_math.cos(thet);
--
        aphi(1,1):=1.0;
        aphi(1,2):=0.0;
        aphi(1,3):=0.0;
        aphi(2,1):=0.0;
        aphi(2,2):= real_math.cos(phi);
        aphi(2,3):=-real_math.sin(phi);
        aphi(3,1):=0.0;
        aphi(3,2):= real_math.sin(phi);
        aphi(3,3):= real_math.cos(phi);
end get_mats;

end mat_math;
```

APPENDIX J

```
cat arcj.bat ast tests.doc ast fs1.in ast fs2.in ast fs3.in >tmp1
cat tmp1 ast fs4a.in ast fs4b.in ast fs5a.in ast fs5b.in >tmp2
cat tmp2 ast fs6a.in ast fs6b.in ast fs7a.in ast fs7b.in >tmp3
cat tmp3 ast fs8.in ast fs4b.out ast fs8.out >tmp
rm tmp1 tmp2 tmp3

**********************************************************************

This file documents several test cases run on flight sim, 8/91.

1.  Straight and level
2.  Straight climb (flight path angle of 7 deg)
3.  Straight glide (flight path angle of -3 deg)
4.  Level 60-deg turn, left then right
5.  Level 30-deg turn, left then right
6.  Climbing (thet=7 deg) 30-deg turn, left then right
7.  Descending (thet=-3 deg) 60-deg turn, left then right
8.  Climbing, turning deceleration

The climb and glide profiles (except #8) impose the constant airspeed
constraint.  All profiles are chosen such that thet ranges from -90 to
90 deg.

FS.EXE imported from \harmon\fs.

**********************************************************************

0.0    50.0001  0.01  100  0 0
0.018   6.0  0.85
500.0   36000.0  2682.2752
10000.0  500.0   0.0 0.0 0.0
0.0 500.0 2
0.336359
0.336359

**********************************************************************

0.0    20.0001  0.01  50  0 1
0.018   6.0  0.85
500.0   36000.0  7058.3452
10000.0  500.0   0.0  0.122173  0.0
0.0 16.5 4
0.333852
0.344979
0.356
0.367

**********************************************************************

0.0    50.0001  0.01  100  0  1
0.018   6.0  0.85
500.0   36000.0  0.0
10000.0  500.0   0.0  -0.052360  0.0
0.0 38.2 4
0.335898
0.325064
0.314
0.303

**********************************************************************
```

```
0.0    60.0001  0.01  100  0  0
0.018   6.0  0.85
500.0   36000.0  4949.5647
10000.0  500.0   0.0  0.0  -1.047198
0.0 500.0 2
0.672718
0.672718

**********************************************************************

0.0    60.0001  0.01  100  0  0
0.018   6.0  0.85
500.0   36000.0  4949.5647
10000.0  500.0   0.0  0.0   1.047198
0.0 500.0 2
0.672718
0.672718

**********************************************************************

0.0    180.0001  0.01  500  0  0
0.018   6.0  0.85
500.0   36000.0  2934.1966
10000.0  500.0   0.0  0.0  -0.523599
0.0 500.0 2
0.388394
0.388394

**********************************************************************

0.0    180.0001  0.01  500  0  0
0.018   6.0  0.85
500.0   36000.0  2934.1966
10000.0  500.0   0.0  0.0   0.523599
0.0 500.0 2
0.388394
0.388394

**********************************************************************

0.0    80.0001  0.01  100  0  1
0.018   6.0  0.85
500.0   36000.0  7306.52
10000.0  500.0   0.0  0.122173  -0.523599
0.0 16.4 6
0.385499
0.398348
0.411625
0.425345
0.439521
0.454172

**********************************************************************

0.0    80.0001  0.01  100  0  1
0.018   6.0  0.85
500.0   36000.0  7306.52
10000.0  500.0   0.0  0.122173   0.523599
0.0 16.4 6
```

```
0.385499
0.398348
0.411625
0.425345
0.439521
0.454172

*************************************************************************

0.0    200.0001  0.01  100  0  1
0.018   6.0  0.85
500.0   36000.0  3057.2
10000.0  500.0   0.0  -0.052360  -1.047198
0.0  76.43  6
0.671796
0.629157
0.589225
0.551827
0.516802
0.484001

*************************************************************************

0.0    200.0001  0.01  100  0  1
0.018   6.0  0.85
500.0   36000.0  3057.2
10000.0  500.0   0.0  -0.052360   1.047198
0.0  76.43  6
0.671796
0.629157
0.589225
0.551827
0.516802
0.484001

*************************************************************************

0.0    35.0001  0.01  25  0  0
0.018   6.0  0.85
500.0   36000.0  7252.38
10000.0  500.0   0.0  0.122173   0.523599
0.0  18.24  3
0.385499
0.491787
0.643164

*************************************************************************

tmin,tmax,dt,iprint,idebug,iconas=    0.0000   60.0001    0.0100 100   0   0
cd0,aspect,eff=  0.0180   6.0000   0.8500
swing,weight,thrust=          500       36000         4949.5647
alt,aspd,psi,thet,phi=        10000      500   0.0000   0.0000   1.0472
mass,rho,qbar=   1118.92      0.00171246      214.0569
    0.00   500.00      2
    0.00   0.672718
  500.00   0.672718
apsi,athet,aphi
      1.00     -0.00      0.00
      0.00      1.00      0.00
      0.00      0.00      1.00
      1.00       0.00       0.00
      0.00       1.00       0.00
     -0.00       0.00       1.00

      1.00       0.00       0.00
      0.00       0.50      -0.87
      0.00       0.87       0.50

aipsi,aithet,aiphi
      1.00       0.00      -0.00
     -0.00       1.00      -0.00
      0.00      -0.00       1.00

      1.00      -0.00       0.00
     -0.00       1.00      -0.00
      0.00      -0.00       1.00

      1.00      -0.00      -0.00
     -0.00       0.50       0.87
      0.00      -0.87       0.50

wt1,wts,xdots,xdot1
      0.00
      0.00
  36000.00

      0.00
  31176.92
  17999.99

    500.00
      0.00
      0.00

    500.00
      0.00
      0.00

Output from program fs, flight sim for unsteady

   time      x      y       z aspd xdot ydot zdot  psi thet  phi  lift  drag

   0.00      0      0  -10000  500  500    0    0    0    0   60 72000  4950
   1.00    499     28  -10000  500  497   56    0    6   -0   60 72000  4950
   2.00    992    112  -10000  500  488  111    0   13   -0   60 72000  4950
   3.00   1472    249  -10000  500  472  164    0   19   -0   60 72000  4950
   4.00   1934    440  -10000  500  451  216    0   26   -0   60 72000  4950
   5.00   2373    680  -10000  500  425  264    0   32   -0   60 72000  4950
   6.00   2781    968  -10000  500  392  310    0   38   -0   60 72000  4950
   7.00   3156   1299  -10000  500  356  352    0   45   -0   60 72000  4950
   8.00   3491   1670  -10000  500  314  389    0   51   -0   60 72000  4950
   9.00   3783   2077  -10000  500  269  422    0   57   -0   60 72000  4950
  10.00   4027   2512  -10000  500  221  449    0   64   -0   60 72000  4950
  11.00   4222   2973  -10000  500  169  471    0   70   -0   60 72000  4950
  12.00   4365   3452  -10000  500  116  487    0   77   -0   60 72000  4950
  13.00   4453   3945  -10000  500   61  497    0   83   -0   60 72000  4950
  14.00   4486   4444  -10000  500    6  500    0   89   -0   60 72000  4950
  15.00   4464   4943  -10000  500  -50  498    0   96   -0   60 72000  4950
  16.00   4386   5437  -10000  500 -105  489    0  102   -0   60 72000  4950
  17.00   4253   5920  -10000  500 -159   475    0  109   -0   60  72000  4950
  18.00   4068   6385  -10000  500 -211   454    0  115   -0   60  72000  4950
  19.00   3833   6826  -10000  500 -260   428    0  121   -0   60  72000  4950
  20.00   3549   7238  -10000  500 -306   396    0  128   -0   60  72000  4950
  21.00   3222   7617  -10000  500 -348   360    0  134   -0   60  72000  4950
  22.00   2855   7957  -10000  500 -386   319    0  140   -0   60  72000  4950
  23.00   2452   8254  -10000  500 -419   274    0  147   -0   60  72000  4950
  24.00   2018   8504  -10000  500 -447   226    0  153   -0   60  72000  4950
  25.00   1560   8705  -10000  500 -469   175    0  160   -0   60  72000  4950
  26.00   1082   8853  -10000  500 -486   122    0  166   -0   60  72000  4950
  27.00    590   8947  -10000  500 -496    67    0  172   -0   60  72000  4950
  28.00     91   8987  -10000  500 -501    12    0  179   -0   60  72000  4950
  29.00   -409   8971  -10000  500 -499   -44    0  185   -0   60  72000  4950
  30.00   -905   8899  -10000  500 -491   -99    0  191   -0   60  72000  4950
  31.00  -1389   8772  -10000  500 -477  -153    0  198   -0   60  72000  4950
  32.00  -1857   8593  -10000  500 -457  -205    0  204   -0   60  72000  4950
  33.00  -2301   8363  -10000  500 -432  -254    0  211   -0   60  72000  4950
  34.00  -2718   8085  -10000  500 -401  -301    0  217   -0   60  72000  4950
  35.00  -3101   7762  -10000  500 -365  -343    0  223   -0   60  72000  4950
  36.00  -3446   7399  -10000  500 -325  -382   -0  230    0   60  72000  4950
  37.00  -3748   7000  -10000  500 -280  -415   -0  236    0   60  72000  4950
  38.00  -4005   6569  -10000  500 -232  -444   -0  242    0   60  72000  4950
  39.00  -4212   6113  -10000  500 -182  -467   -0  249    0   60  72000  4950
  40.00  -4367   5637  -10000  500 -129  -484   -0  255    0   60  72000  4950
  41.00  -4468   5146  -10000  500  -74  -496   -0  261    0   60  72000  4950
  42.00  -4514   4647  -10000  500  -19  -501   -0  268    0   60  72000  4950
  43.00  -4505   4146  -10000  500   37  -500   -0  274    0   60  72000  4950
  44.00  -4440   3649  -10000  500   92  -493   -0  281    0   60  72000  4950
  45.00  -4320   3163  -10000  500  146  -480   -0  287    0   60  72000  4950
  46.00  -4147   2692  -10000  500  199  -460   -0  293    0   60  72000  4950
  47.00  -3923   2244  -10000  500  248  -436   -0  300    0   60  72000  4950
  48.00  -3651   1823  -10000  500  295  -405   -0  306    0   60  72000  4950
  49.00  -3334   1435  -10000  500  338  -370   -0  312    0   60  72000  4950
  50.00  -2975   1085  -10000  500  377  -330   -0  319    0   60  72000  4950
  51.00  -2580    776  -10000  500  412  -287   -0  325    0   60  72000  4950
  52.00  -2153    513  -10000  500  441  -239   -0  332    0   60  72000  4950
  53.00  -1700    300  -10000  500  465  -189   -0  338    0   60  72000  4950
  54.00  -1225    137  -10000  500  483  -136   -0  344    0   60  72000  4950
  55.00   -736     28  -10000  500  495   -82   -0  351    0   60  72000  4950
  56.00   -237    -25  -10000  500  501   -26   -0  357    0   60  72000  4950
  57.00    264    -24  -10000  500  501    29   -0    3    0   60  72000  4950
  58.00    762     34  -10000  500  495    85   -0   10    0   60  72000  4950
  59.00   1251    146  -10000  500  482   139   -0   16    0   60  72000  4950
  60.00   1725    312  -10000  500  464   192   -0   22    0   60  72000  4950

**********************************************************************

tmin,tmax,dt,iprint,idebug,iconas=     0.0000   35.0001    0.0100  25    0    0
cd0,aspect,eff=  0.0180   6.0000   0.8500
swing,weight,thrust=        500        36000         7252.3800
alt,aspd,psi,thet,phi=      10000       500    0.0000    0.1222    0.5236
mass,rho,qbar=   1118.92     0.00171246      214.0569
    0.00    18.24        3
    0.00   0.385499
   18.24   0.491787
   36.48   0.643164
apsi,athet,aphi
      1.00     -0.00       0.00
      0.00      1.00       0.00
      0.00      0.00       1.00
      0.99      0.00      0.12
      0.00      1.00      0.00
     -0.12      0.00      0.99

      1.00      0.00      0.00
      0.00      0.87     -0.50
      0.00      0.50      0.87

aipsi,aithet,aiphi
      1.00      0.00     -0.00
     -0.00      1.00     -0.00
      0.00     -0.00      1.00

      0.99     -0.00     -0.12
     -0.00      1.00     -0.00
      0.12     -0.00      0.99

      1.00     -0.00     -0.00
     -0.00      0.87      0.50
      0.00     -0.50      0.87

wt1,wts,xdots,xdot1
      0.00
      0.00
  36000.00

  -4387.29
  17865.84
  30944.52

    500.00
      0.00
      0.00

    496.27
      0.00
    -60.93

Output from program fs, flight sim for unsteady

   time       x      y        z aspd xdot ydot zdot  psi thet  phi   lift  drag

   0.00       0      0   -10000  500  496    0  -61    0    7   30  41259  2919
   0.25     124      1   -10015  500  496    5  -61    1    7   30  41387  2925
   0.50     248      2   -10030  500  496    9  -61    1    7   30  41520  2931
   0.75     372      5   -10046  500  496   14  -61    2    7   30  41652  2937
   1.00     496      9   -10061  500  496   19  -61    2    7   30  41784  2943
   1.25     620     15   -10076  500  496   23  -61    3    7   30  41914  2948
   1.50     744     21   -10092  500  495   28  -61    3    7   30  42045  2954
   1.75     868     29   -10107  500  495   33  -62    4    7   30  42174  2960
   2.00     991     37   -10122  500  495   37  -62    4    7   30  42302  2966
   2.25    1115     47   -10138  500  494   42  -62    5    7   30  42429  2972
   2.50    1238     58   -10153  500  494   47  -62    5    7   30  42556  2977
   2.75    1362     71   -10169  500  493   51  -62    6    7   30  42681  2983
   3.00    1485     84   -10185  500  493   56  -63    7    7   30  42804  2989
   3.25    1608     99   -10200  500  492   61  -63    7    7   30  42927  2994
   3.50    1731    115   -10216  500  491   66  -63    8    7   30  43048  3000
   3.75    1854    132   -10232  499  490   70  -64    8    7   30  43167  3005
   4.00    1976    150   -10248  499  490   75  -64    9    7   30  43285  3010
```

4.25	2098	169	-10264	499	489	80	-64	9	7	30	43401	3016
4.50	2220	190	-10280	499	488	85	-65	10	7	30	43516	3021
4.75	2342	212	-10296	499	487	89	-65	10	8	30	43628	3026
5.00	2464	235	-10313	499	486	94	-66	11	8	30	43739	3031
5.25	2585	259	-10329	499	484	99	-66	12	8	30	43847	3036
5.50	2706	284	-10346	499	483	104	-67	12	8	30	43954	3040
5.75	2827	311	-10363	499	482	109	-67	13	8	30	44058	3045
6.00	2947	339	-10380	498	481	113	-68	13	8	30	44161	3050
6.25	3067	368	-10397	498	479	118	-68	14	8	30	44260	3054
6.50	3186	398	-10414	498	478	123	-69	14	8	30	44358	3058
6.75	3306	429	-10431	498	476	127	-70	15	8	30	44453	3062
7.00	3425	462	-10449	498	475	132	-70	16	8	30	44545	3066
7.25	3543	495	-10466	498	473	137	-71	16	8	30	44635	3070
7.50	3661	530	-10484	497	471	142	-71	17	8	30	44723	3074
7.75	3779	566	-10502	497	470	146	-72	17	8	30	44807	3077
8.00	3896	603	-10520	497	468	151	-73	18	8	30	44889	3081
8.25	4013	642	-10538	497	466	156	-73	18	9	30	44967	3084
8.50	4129	681	-10557	496	464	160	-74	19	9	30	45043	3087
8.75	4244	722	-10575	496	462	165	-75	20	9	30	45116	3090
9.00	4360	764	-10594	496	460	170	-76	20	9	30	45186	3092
9.25	4474	807	-10613	495	458	174	-76	21	9	30	45252	3095
9.50	4588	851	-10633	495	455	179	-77	21	9	30	45316	3097
9.75	4702	896	-10652	495	453	183	-78	22	9	30	45376	3099
10.00	4815	942	-10671	494	451	188	-79	23	9	30	45433	3101
10.25	4927	990	-10691	494	448	192	-79	23	9	30	45486	3103
10.50	5039	1038	-10711	494	446	197	-80	24	9	30	45537	3105
10.75	5150	1088	-10731	493	443	201	-81	24	9	30	45583	3106
11.00	5260	1139	-10752	493	441	205	-82	25	10	30	45627	3107
11.25	5370	1191	-10772	493	438	210	-83	26	10	30	45666	3108
11.50	5479	1244	-10793	492	435	214	-84	26	10	30	45702	3109
11.75	5588	1298	-10814	492	432	219	-84	27	10	30	45735	3109
12.00	5695	1353	-10835	491	429	223	-85	27	10	30	45764	3110
12.25	5802	1410	-10857	491	426	227	-86	28	10	30	45789	3110
12.50	5909	1467	-10878	490	423	231	-87	29	10	30	45811	3110
12.75	6014	1525	-10900	490	420	235	-88	29	10	30	45829	3109
13.00	6119	1585	-10922	489	417	239	-89	30	10	30	45843	3109
13.25	6223	1645	-10944	489	414	244	-89	30	11	30	45853	3108
13.50	6326	1706	-10967	488	411	248	-90	31	11	30	45860	3107
13.75	6428	1769	-10990	488	408	252	-91	32	11	30	45862	3106
14.00	6530	1832	-11013	487	404	256	-92	32	11	30	45861	3104
14.25	6630	1897	-11036	486	401	259	-93	33	11	30	45857	3103
14.50	6730	1962	-11059	486	397	263	-94	34	11	30	45848	3101
14.75	6829	2028	-11082	485	394	267	-95	34	11	30	45836	3099
15.00	6927	2096	-11106	484	390	271	-95	35	11	30	45819	3096
15.25	7024	2164	-11130	484	387	275	-96	35	11	30	45799	3094
15.50	7120	2233	-11154	483	383	278	-97	36	12	30	45775	3091
15.75	7215	2303	-11179	482	379	282	-98	37	12	30	45747	3088
16.00	7310	2374	-11203	482	375	285	-99	37	12	30	45716	3085
16.25	7403	2446	-11228	481	372	289	-99	38	12	30	45681	3081
16.50	7495	2518	-11253	480	368	292	-100	38	12	30	45642	3078
16.75	7587	2592	-11278	479	364	296	-101	39	12	30	45599	3074
17.00	7677	2666	-11303	479	360	299	-102	40	12	30	45552	3070
17.25	7767	2741	-11329	478	356	302	-103	40	12	30	45502	3066
17.50	7855	2817	-11355	477	352	305	-103	41	12	30	45448	3061
17.75	7943	2894	-11381	476	348	309	-104	42	13	30	45391	3056
18.00	8029	2972	-11407	475	344	312	-105	42	13	30	45330	3051
18.25	8114	3050	-11433	475	339	315	-105	43	13	30	45265	3046
18.50	8199	3129	-11459	474	335	318	-106	43	13	30	45254	3044
18.75	8282	3209	-11486	473	331	321	-107	44	13	30	45238	3042
19.00	8364	3289	-11513	472	327	324	-108	45	13	30	45218	3040

19.25	8445	3371	-11540	471	322	326	-108	45	13	30	45194	3037
19.50	8525	3453	-11567	470	318	329	-109	46	13	30	45166	3035
19.75	8604	3535	-11594	469	313	332	-110	47	14	30	45133	3032
20.00	8682	3618	-11622	468	309	334	-110	47	14	30	45097	3028
20.25	8758	3702	-11649	467	305	337	-111	48	14	30	45056	3025
20.50	8834	3787	-11677	466	300	339	-112	49	14	30	45011	3021
20.75	8908	3872	-11705	465	295	342	-112	49	14	30	44963	3017
21.00	8982	3958	-11733	464	291	344	-113	50	14	30	44910	3013
21.25	9054	4044	-11762	463	286	347	-113	50	14	30	44853	3009
21.50	9125	4131	-11790	462	282	349	-114	51	14	30	44792	3004
21.75	9194	4219	-11819	461	277	351	-115	52	14	30	44727	3000
22.00	9263	4307	-11847	460	272	353	-115	52	14	30	44659	2995
22.25	9331	4395	-11876	459	268	355	-116	53	15	30	44586	2990
22.50	9397	4484	-11905	458	263	357	-116	54	15	30	44510	2984
22.75	9462	4574	-11934	457	258	359	-117	54	15	30	44430	2979
23.00	9526	4664	-11964	456	253	361	-117	55	15	30	44346	2973
23.25	9588	4754	-11993	455	248	362	-118	56	15	30	44259	2967
23.50	9650	4845	-12023	454	244	364	-118	56	15	30	44168	2961
23.75	9710	4936	-12052	453	239	366	-119	57	15	30	44073	2954
24.00	9769	5028	-12082	451	234	367	-119	58	15	30	43975	2948
24.25	9827	5120	-12112	450	229	369	-120	58	15	30	43874	2941
24.50	9884	5212	-12142	449	224	370	-120	59	15	30	43770	2934
24.75	9939	5305	-12172	448	219	372	-120	59	16	30	43662	2927
25.00	9993	5398	-12202	447	214	373	-121	60	16	30	43551	2920
25.25	10046	5492	-12232	446	210	374	-121	61	16	30	43437	2913
25.50	10098	5585	-12262	444	205	375	-121	61	16	30	43320	2905
25.75	10149	5679	-12293	443	200	376	-121	62	16	30	43200	2897
26.00	10198	5773	-12323	442	195	377	-122	63	16	30	43077	2889
26.25	10246	5868	-12353	441	190	378	-122	63	16	30	42951	2881
26.50	10293	5963	-12384	439	185	379	-122	64	16	30	42823	2873
26.75	10339	6058	-12415	438	180	380	-122	65	16	30	42693	2865
27.00	10383	6153	-12445	437	175	381	-122	65	16	30	42560	2857
27.25	10426	6248	-12476	436	170	382	-123	66	16	30	42424	2848
27.50	10468	6344	-12506	434	166	382	-123	67	16	30	42287	2839
27.75	10509	6439	-12537	433	161	383	-123	67	16	30	42147	2831
28.00	10548	6535	-12568	432	156	383	-123	68	17	30	42005	2822
28.25	10587	6631	-12599	430	151	384	-123	69	17	30	41862	2813
28.50	10624	6727	-12629	429	146	384	-123	69	17	30	41716	2804
28.75	10660	6823	-12660	428	141	385	-123	70	17	30	41569	2794
29.00	10694	6919	-12691	426	136	385	-123	70	17	30	41420	2785
29.25	10728	7016	-12721	425	132	385	-123	71	17	30	41270	2776
29.50	10760	7112	-12752	424	127	385	-123	72	17	30	41119	2767
29.75	10791	7208	-12783	422	122	386	-122	72	17	30	40966	2757
30.00	10821	7305	-12813	421	117	386	-122	73	17	30	40813	2748
30.25	10850	7401	-12844	420	113	386	-122	74	17	30	40658	2738
30.50	10877	7498	-12874	418	108	386	-122	74	17	30	40502	2729
30.75	10904	7594	-12904	417	103	386	-121	75	17	30	40346	2719
31.00	10929	7691	-12935	416	98	386	-121	76	17	30	40189	2710
31.25	10953	7787	-12965	414	94	385	-121	76	17	30	40032	2700
31.50	10976	7883	-12995	413	89	385	-120	77	17	30	39874	2690
31.75	10997	7980	-13025	412	85	385	-120	78	17	30	39717	2681
32.00	11018	8076	-13055	411	80	385	-120	78	17	30	39559	2671
32.25	11037	8172	-13085	409	75	384	-119	79	17	30	39401	2662
32.50	11056	8268	-13115	408	71	384	-119	80	17	30	39243	2652
32.75	11073	8364	-13145	407	66	383	-118	80	17	30	39086	2643
33.00	11089	8460	-13174	405	62	383	-118	81	17	30	38929	2633
33.25	11104	8555	-13203	404	57	382	-117	81	17	30	38772	2624
33.50	11117	8651	-13233	403	53	382	-117	82	17	30	38616	2614
33.75	11130	8746	-13262	401	48	381	-116	83	17	30	38461	2605
34.00	11142	8841	-13291	400	44	381	-115	83	17	30	38307	2596